Eine Einführung in Theoretische Informatik

Dennis Komm

10.02.2015

Die zentralen Fragen sind ...

- Was kann man mit einem Computer **nicht** machen?
- **Wie gut** kann man machen, was gemacht werden kann?

Teil 1 – Probleme in Mathe

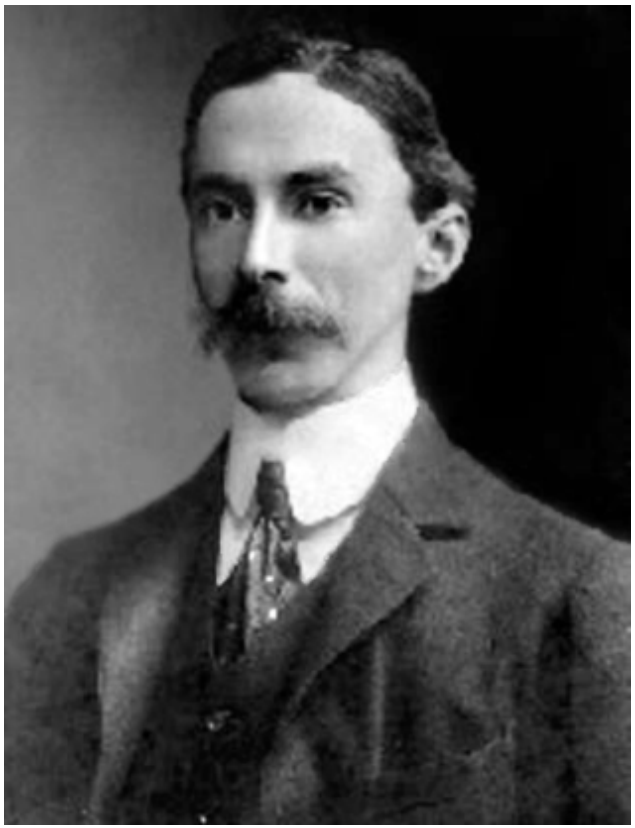
Teil 1 – Probleme in Mathe

Gottfried Wilhelm Leibniz (1646 – 1716)

Wikimedia, Creative Commons

"Meine Herren, lassen Sie uns rechnen."

Bertrand Russell (1872 – 1970)



Wikimedia, Creative Commons

- Sei S die Menge aller Mengen, die sich nicht selbst enthalten, also $S = \{X \mid X \text{ ist eine Menge und } X \notin X\}$
- Enthält S sich selber, also ist $S \in S$ oder $S \notin S$?
- Dies führt zu einem Widerspruch
- Also kann S nicht existieren

Bertrand Russell (1872 – 1970)

Angenommen, es gilt $S \in S$

Bertrand Russell (1872 – 1970)

Angenommen, es gilt $S \in S$

- ⇨ S ist also eine Menge, die sich selbst enthält
- ⇨ S kann nicht in S sein, denn S ist genau so definiert, dass es keine Menge enthält, die sich selbst enthält

Bertrand Russell (1872 – 1970)

Angenommen, es gilt $S \in S$

- $\Rightarrow$ S ist also eine Menge, die sich selbst enthält
- $\Rightarrow$ S kann nicht in S sein, denn S ist genau so definiert, dass es keine Menge enthält, die sich selbst enthält

Na gut, dann nehmen wir eben an, es gilt $S \notin S$

Bertrand Russell (1872 – 1970)

Angenommen, es gilt $S \in S$

- ⇨ S ist also eine Menge, die sich selbst enthält
- ⇨ S kann nicht in S sein, denn S ist genau so definiert, dass es keine Menge enthält, die sich selbst enthält

Na gut, dann nehmen wir eben an, es gilt $S \notin S$

- ⇨ S ist also eine Menge, die sich nicht selbst enthält
- ⇨ S muss in S sein, denn S ist genau so definiert

Bertrand Russell (1872 – 1970)

Angenommen, es gilt $S \in S$

- ⇨ S ist also eine Menge, die sich selbst enthält
- ⇨ S kann nicht in S sein, denn S ist genau so definiert, dass es keine Menge enthält, die sich selbst enthält

Na gut, dann nehmen wir eben an, es gilt $S \notin S$

- ⇨ S ist also eine Menge, die sich nicht selbst enthält
- ⇨ S muss in S sein, denn S ist genau so definiert

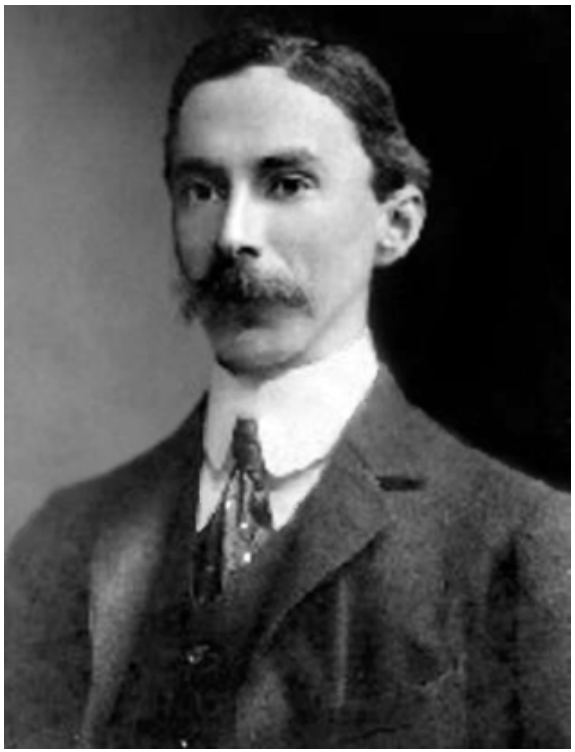
David Hilbert (1862 – 1943)

Wikimedia, Creative Commons

Hilbert-Programm

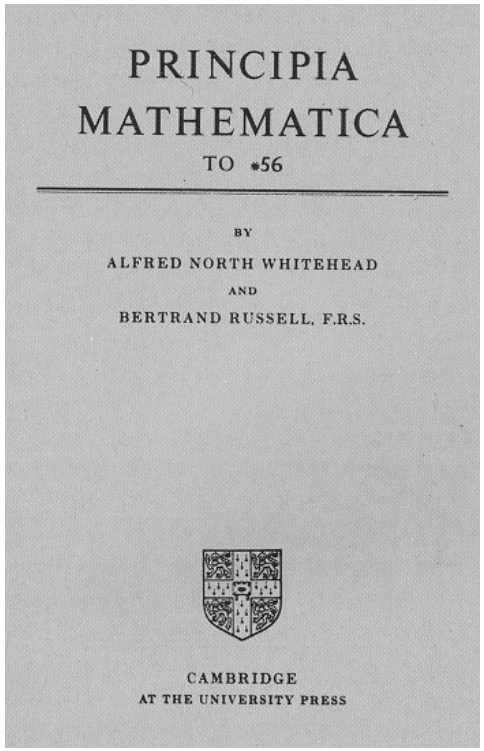
Erstelle ein System von Axiomen und Regeln, in dem jede wahre Aussage bewiesen werden kann und zwar so, dass dabei nie ein Widerspruch entsteht

B. Russell, Alfred N. Whitehead (1861 – 1947)



Wikimedia, Creative Commons

Unbekannter Urheber



Wikimedia, Creative Commons

Kurt Gödel (1862 – 1943)

Wikimedia, Creative Commons

Gödels Unvollständigkeitssatz

Wenn Widerspruchsfreiheit herrscht, existieren wahre Aussagen, die nicht bewiesen werden können

Fazit

Wir wissen heute (dank Gödel) ...

- Die Mathematik ist unvollständig
- Es gibt Aussagen, die weder beweisbar noch widerlegbar sind
- Das wird so bleiben

Fazit

Wir wissen heute (dank Gödel) ...

- Die Mathematik ist unvollständig
- Es gibt Aussagen, die weder beweisbar noch widerlegbar sind
- Das wird so bleiben

Aber was ist mit beweisbaren (widerlegbaren) Aussagen?

- Wie schwer ist es, diese zu beweisen (widerlegen)?
- Kann dies automatisiert werden?

Alan M. Turing (1912 – 1954)

Wikimedia, Creative Commons

Es gibt beweisbare Aussagen, die wir nicht automatisch beantworten lassen können

Die Idee ist der von Gödels Unvollständigkeitssatz sehr ähnlich

Teil 2 – Unendlich mal Unendlich

Teil 2 – Unendlich mal Unendlich

Walt Disney Pictures

- Ein Schäfer hat schwarze und weisse Schafe
- Hat er mehr weisse oder schwarze?
- Er kann nur bis drei zählen

- Ein Schäfer hat schwarze und weisse Schafe
- Hat er mehr weisse oder schwarze?
- Er kann nur bis drei zählen

⇨ Führe jeweils ein weisses und ein schwarzes auf andere Weide

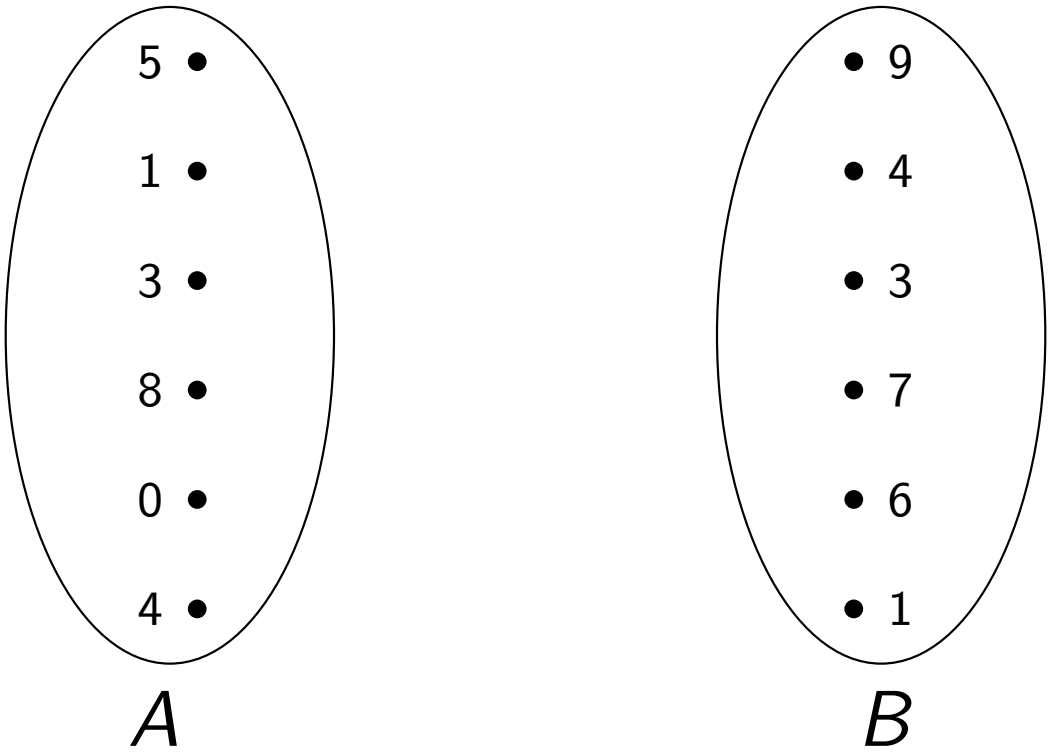
Juraj Hromkovič, *Berechenbarkeit*, Vieweg Teubner

Satz (von Cantor und Bernstein)

Zwei Mengen A und B sind gleich gross, wenn es eine Bijektion zwischen ihnen gibt □

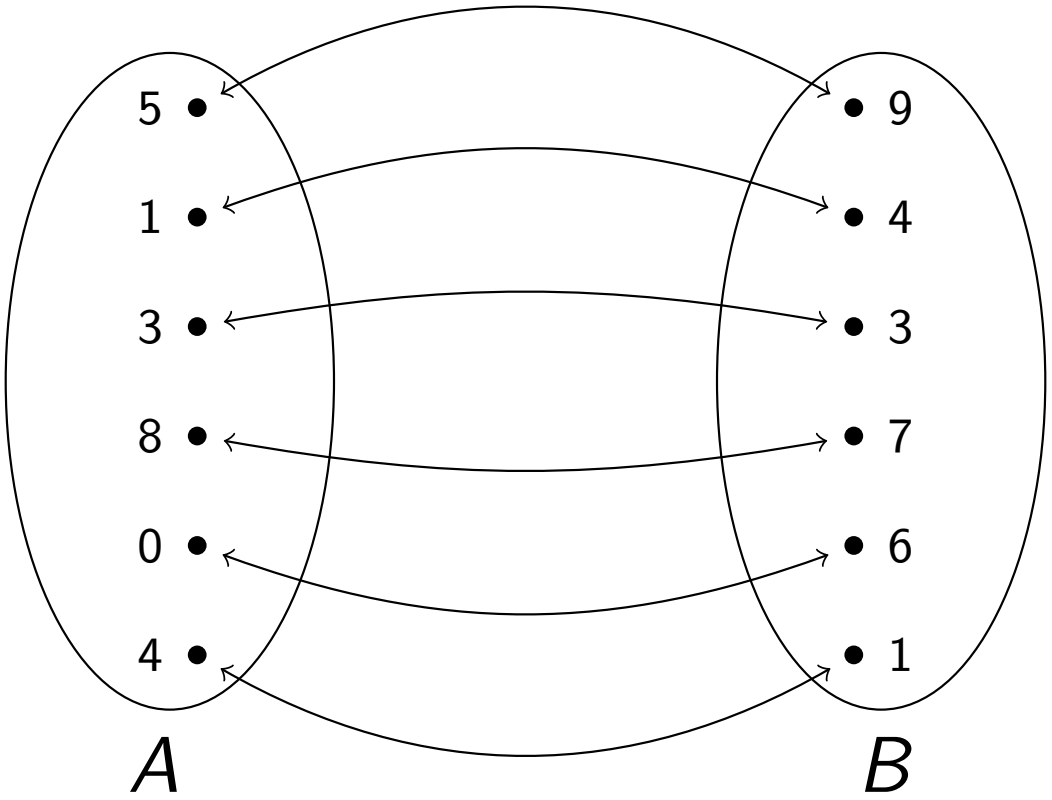
Satz (von Cantor und Bernstein)

Zwei Mengen A und B sind gleich gross, wenn es eine Bijektion zwischen ihnen gibt □



Satz (von Cantor und Bernstein)

Zwei Mengen A und B sind gleich gross, wenn es eine Bijektion zwischen ihnen gibt □

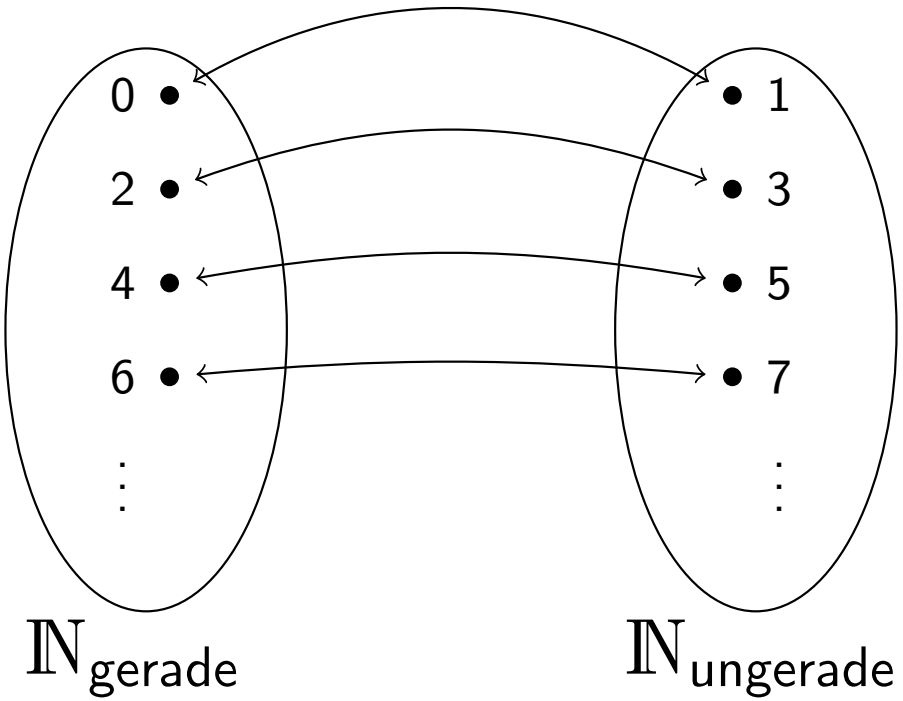


Das gilt auch für unendliche Mengen

- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ (natürliche Zahlen)
- $\mathbb{N}_{\text{gerade}} = \{0, 2, 4, 6, \dots\}$ (gerade Zahlen)
- $\mathbb{N}_{\text{ungerade}} = \{1, 3, 5, 7, \dots\}$ (ungerade Zahlen)

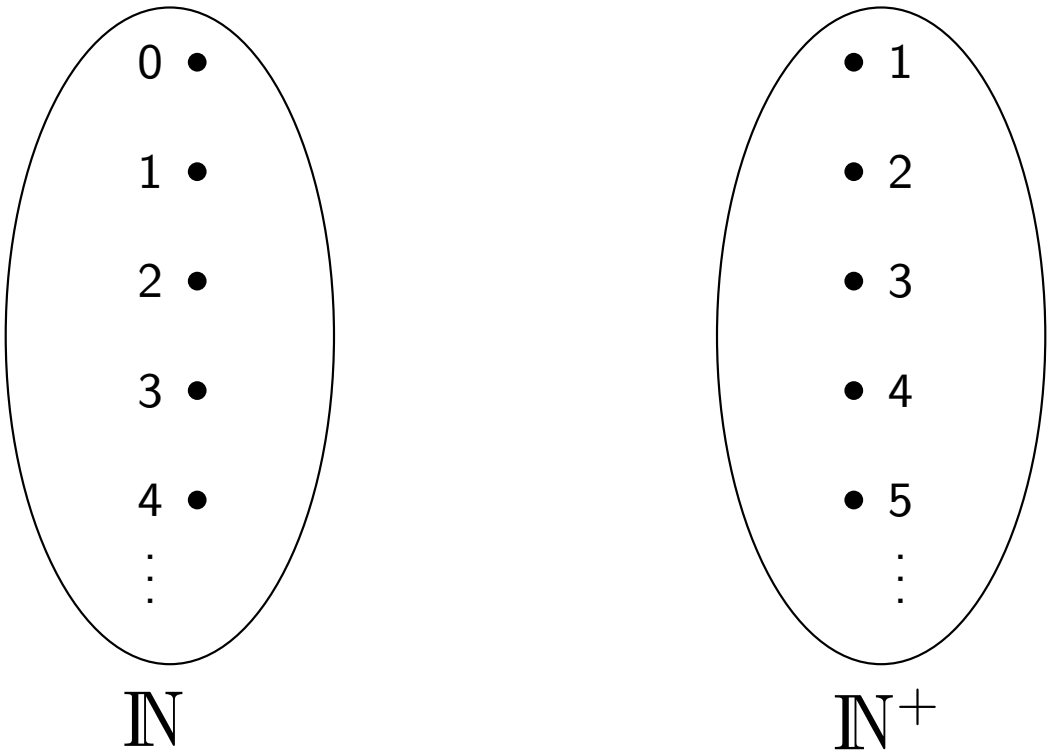
Das gilt auch für unendliche Mengen

- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ (natürliche Zahlen)
- $\mathbb{N}_{\text{gerade}} = \{0, 2, 4, 6, \dots\}$ (gerade Zahlen)
- $\mathbb{N}_{\text{ungerade}} = \{1, 3, 5, 7, \dots\}$ (ungerade Zahlen)



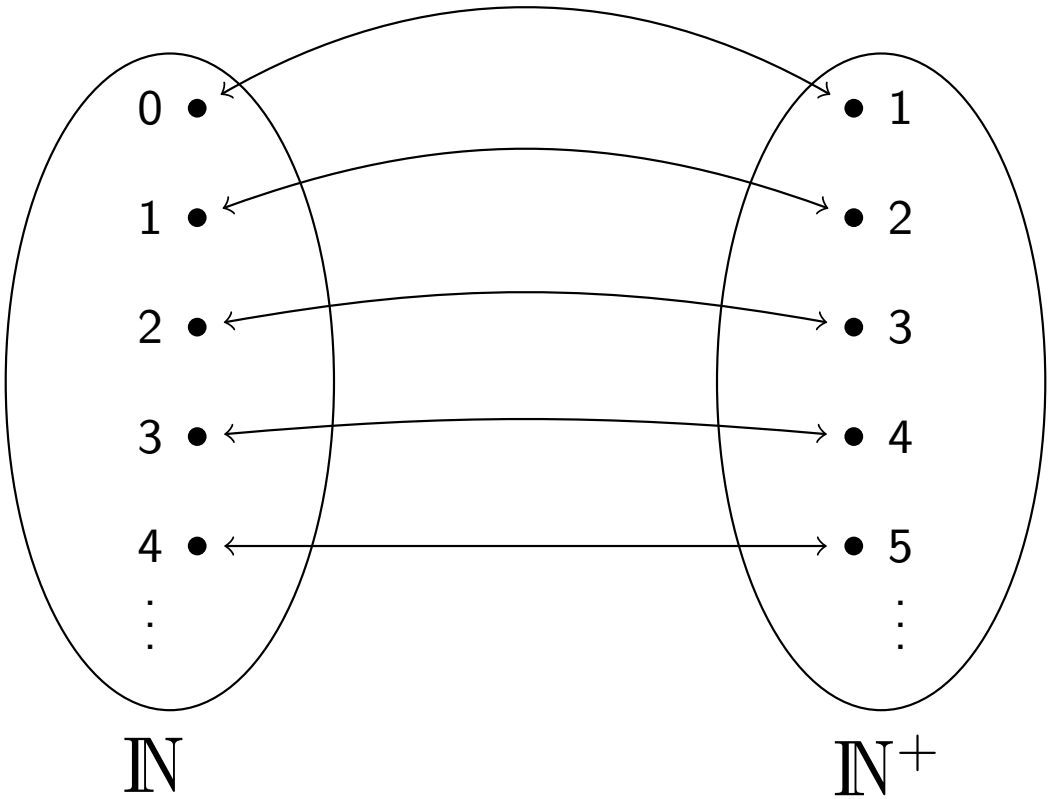
- ⇨ Es gilt also $|\mathbb{N}_{\text{gerade}}| = |\mathbb{N}_{\text{ungerade}}|$
  - Was ist mit $\mathbb{N}^+ = \{1, 2, 3, \dots\}$? (positive natürliche Zahlen)
  - Intuitiv sollte $|\mathbb{N}^+| < |\mathbb{N}|$ sein

- ⇨ Es gilt also $|\mathbb{N}_{\text{gerade}}| = |\mathbb{N}_{\text{ungerade}}|$
  - Was ist mit $\mathbb{N}^+ = \{1, 2, 3, \dots\}$? (positive natürliche Zahlen)
  - Intuitiv sollte $|\mathbb{N}^+| < |\mathbb{N}|$ sein



⇨ Es gilt also $|\mathbb{N}_{\text{gerade}}| = |\mathbb{N}_{\text{ungerade}}|$

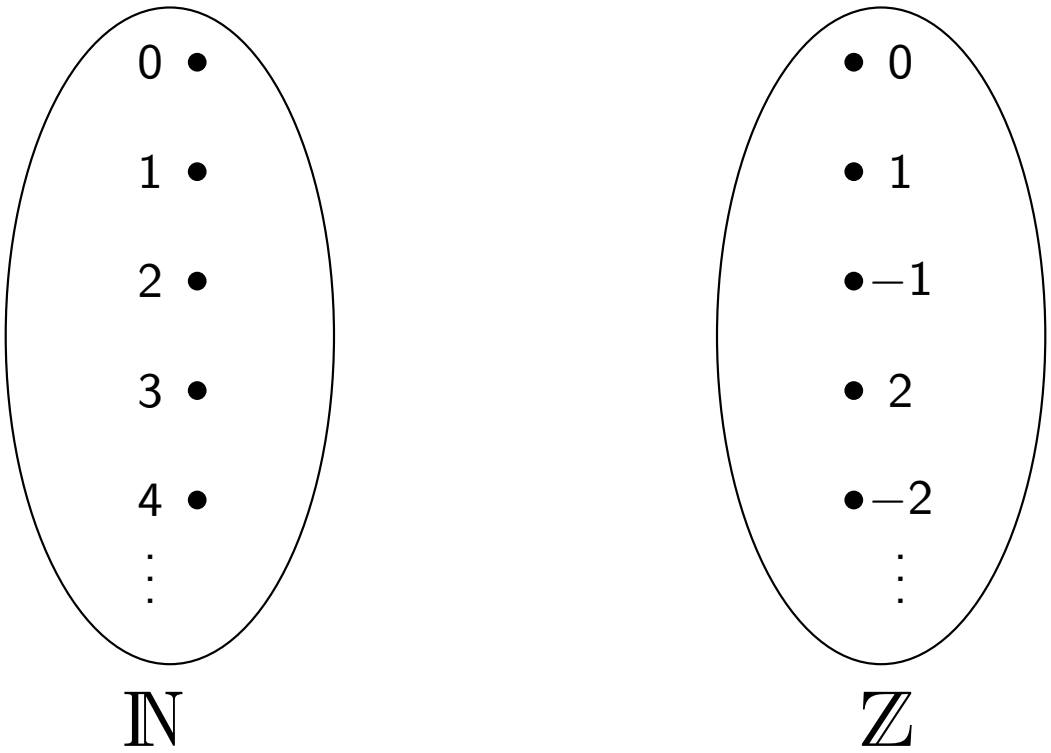
- Was ist mit $\mathbb{N}^+ = \{1, 2, 3, \dots\}$? (positive natürliche Zahlen)
- ~~Intuitiv sollte $|\mathbb{N}^+| < |\mathbb{N}|$ sein~~; es gilt $|\mathbb{N}^+| = |\mathbb{N}|$



⇨ Es gilt ausserdem $|\mathbb{N}^+| = |\mathbb{N}|$

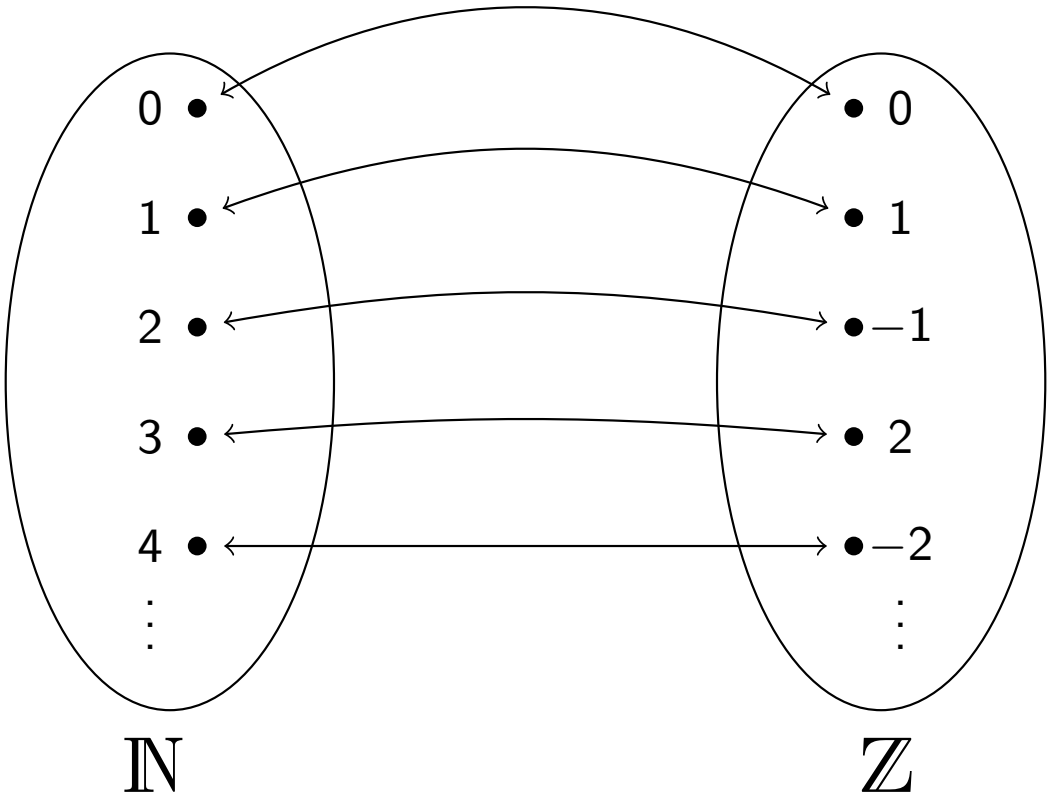
- Was ist mit $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$? (ganze Zahlen)
- Intuitiv sollte $|\mathbb{Z}| = 2 \cdot |\mathbb{N}|$ sein

- ⇨ Es gilt ausserdem $|\mathbb{N}^+| = |\mathbb{N}|$
  - Was ist mit $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$? (ganze Zahlen)
  - Intuitiv sollte $|\mathbb{Z}| = 2 \cdot |\mathbb{N}|$ sein



⇨ Es gilt ausserdem $|\mathbb{N}^+| = |\mathbb{N}|$

- Was ist mit $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$? (ganze Zahlen)
- ~~Intuitiv sollte $|\mathbb{Z}| = 2 \cdot |\mathbb{N}|$ sein~~; es gilt $|\mathbb{Z}| = |\mathbb{N}|$



Definition

Eine unendliche Menge A heisst **abzählbar**, wenn sie genau so gross ist wie $\mathbb{N}$

Definition

Eine unendliche Menge A heisst **abzählbar**, wenn sie genau so gross ist wie $\mathbb{N}$

⇨ Jedem Element aus A wird eine natürliche Zahl zugeordnet

Definition

Eine unendliche Menge A heisst **abzählbar**, wenn sie genau so gross ist wie $\mathbb{N}$

⇨ Jedem Element aus A wird eine natürliche Zahl zugeordnet

Satz

$\mathbb{Z}$ ist abzählbar

Definition

Eine unendliche Menge A heisst **abzählbar**, wenn sie genau so gross ist wie $\mathbb{N}$

⇨ Jedem Element aus A wird eine natürliche Zahl zugeordnet

Satz

$\mathbb{Z}$ ist abzählbar

- $\mathbb{Q}^+ = \{\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \ldots\}$ (positive rationale Zahlen)

Definition

Eine unendliche Menge A heisst **abzählbar**, wenn sie genau so gross ist wie $\mathbb{N}$

⇨ Jedem Element aus A wird eine natürliche Zahl zugeordnet

Satz

$\mathbb{Z}$ ist abzählbar

- $\mathbb{Q}^+ = \{\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \ldots\}$ (positive rationale Zahlen)

Satz

$\mathbb{Q}^+$ ist abzählbar

Definition

Eine unendliche Menge A heisst **abzählbar**, wenn sie genau so gross ist wie $\mathbb{N}$

⇨ Jedem Element aus A wird eine natürliche Zahl zugeordnet

Satz

$\mathbb{Z}$ ist abzählbar

- $\mathbb{Q}^+ = \{\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \ldots\}$ (positive rationale Zahlen)

Satz

$\mathbb{Q}^+$ ist abzählbar

Satz

$\mathbb{Q}$ ist abzählbar

David Hilbert (1862 – 1943)

Wikimedia, Creative Commons

Das Hilbert-Hotel

- Unendlich viele Räume
- In jedem Raum logiert ein Gast

David Hilbert (1862 – 1943)

Wikimedia, Creative Commons

Das Hilbert-Hotel

- Unendlich viele Räume
- In jedem Raum logiert ein Gast
- Ein neuer Gast kommt

David Hilbert (1862 – 1943)

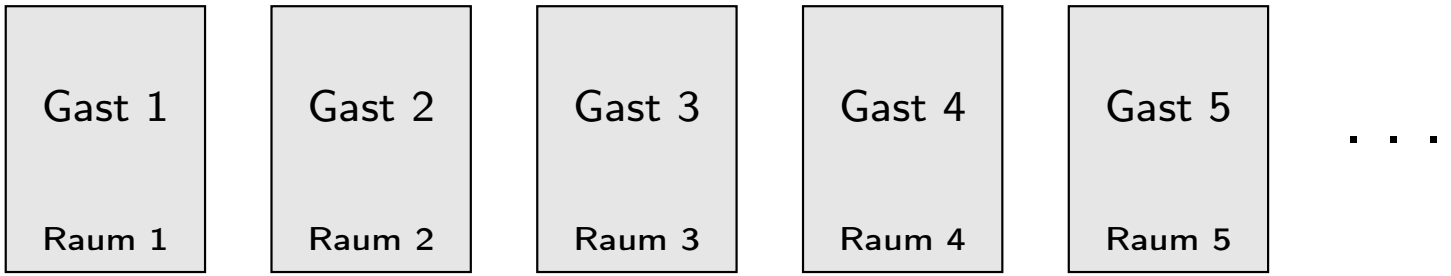
Wikimedia, Creative Commons

Das Hilbert-Hotel

- Unendlich viele Räume
- In jedem Raum logiert ein Gast
- Ein neuer Gast kommt
- Können wir ihm einen Raum zuweisen ohne jemanden rauszuschmeissen?

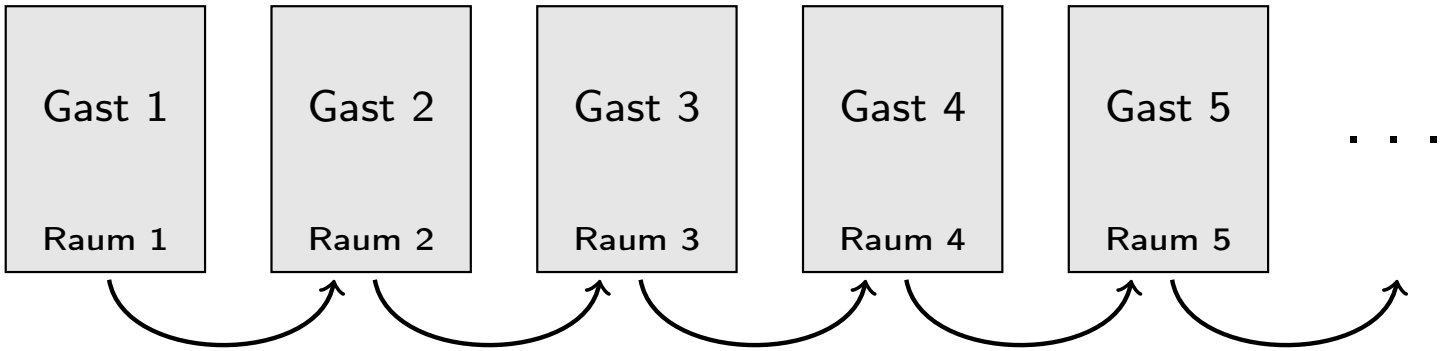
David Hilbert (1862 – 1943)

Ein neuer Gast kommt . . .



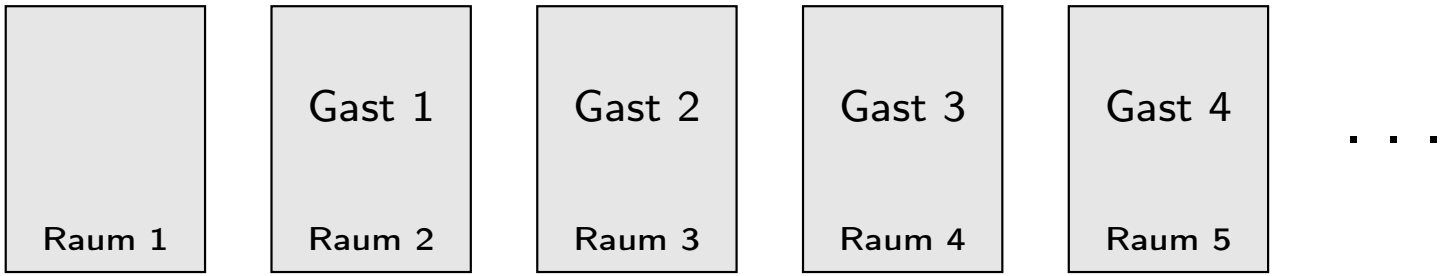
David Hilbert (1862 – 1943)

Ein neuer Gast kommt . . .



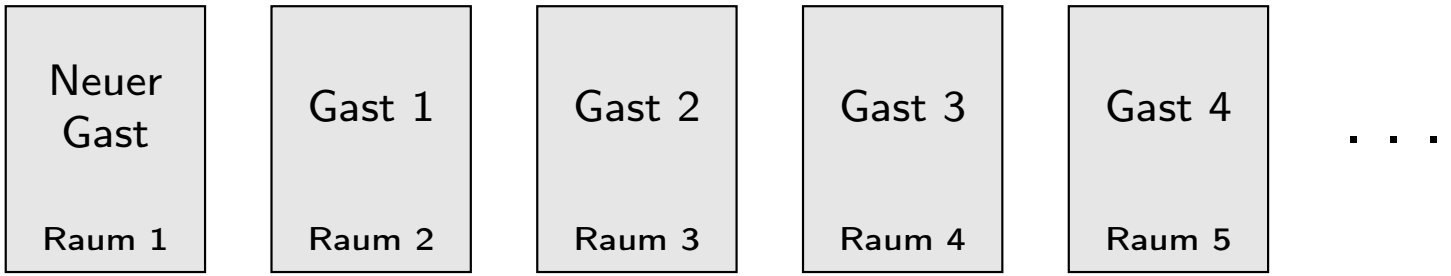
David Hilbert (1862 – 1943)

Ein neuer Gast kommt . . .



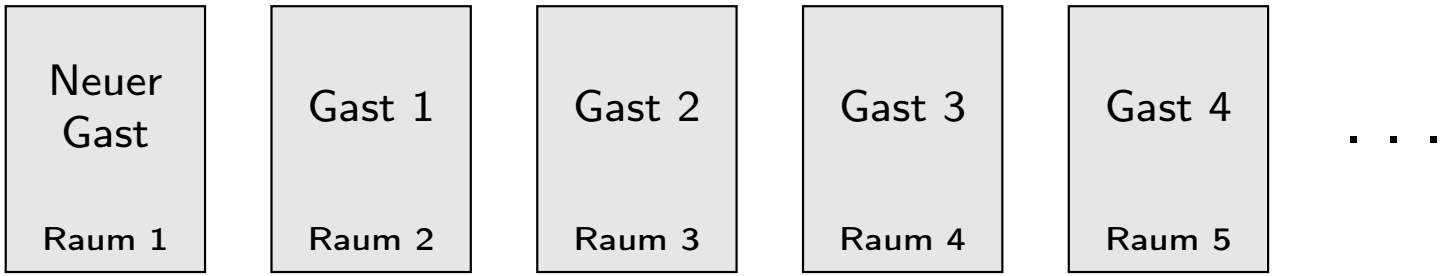
David Hilbert (1862 – 1943)

Ein neuer Gast kommt . . .

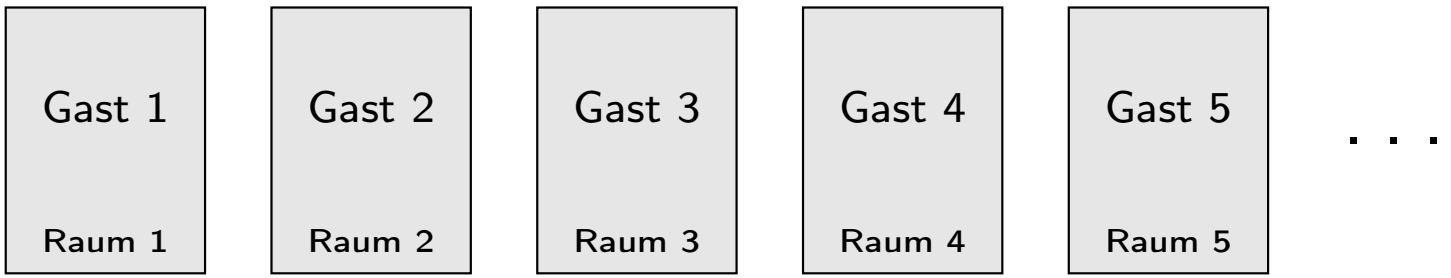


David Hilbert (1862 – 1943)

Ein neuer Gast kommt . . .

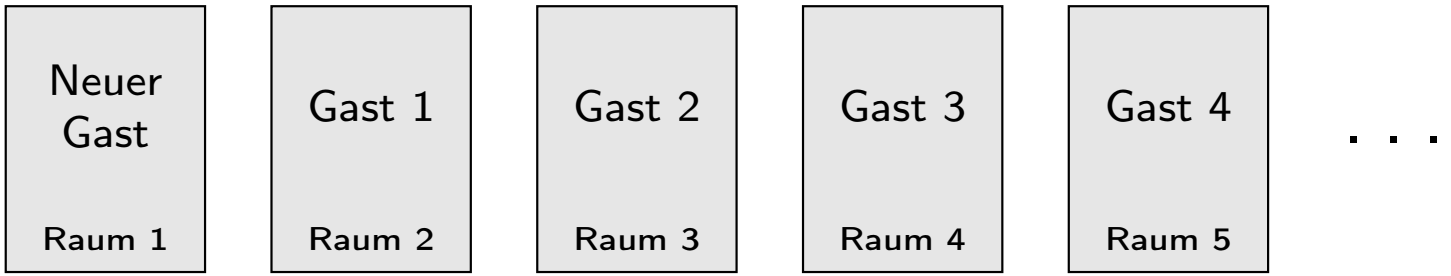


Unendlich viele neue Gäste kommen . . .

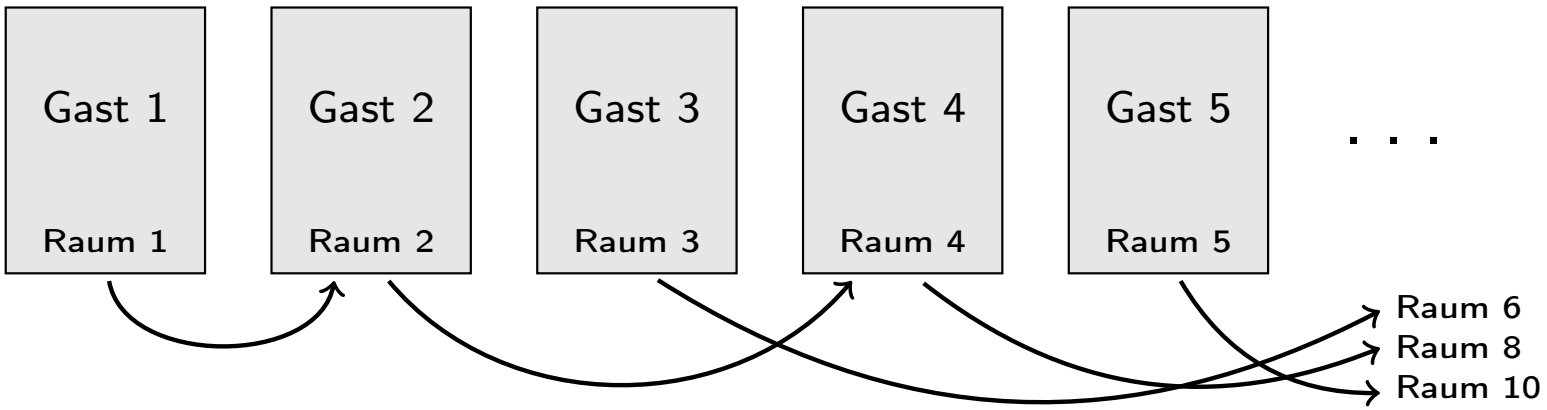


David Hilbert (1862 – 1943)

Ein neuer Gast kommt . . .

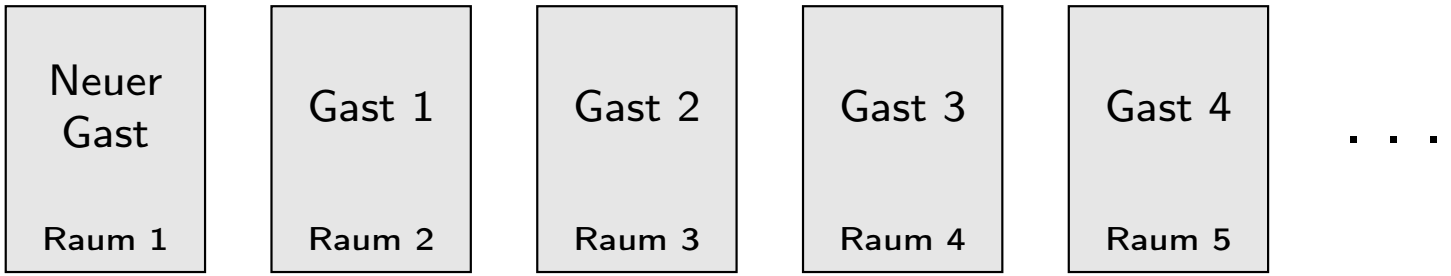


Unendlich viele neue Gäste kommen . . .

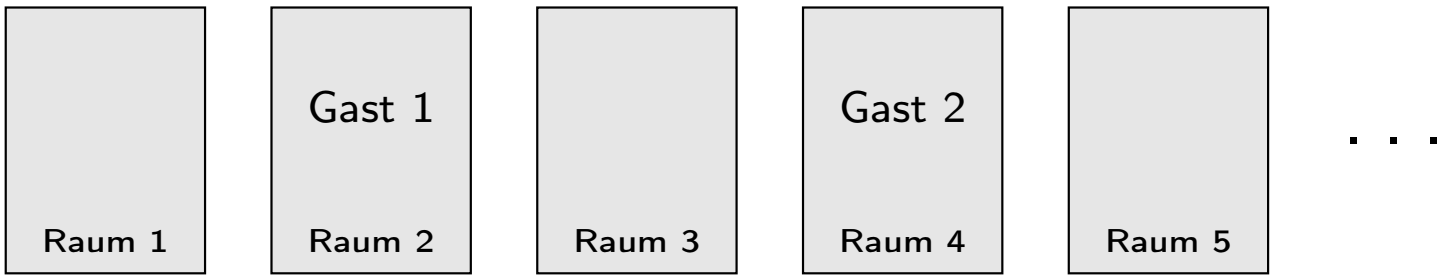


David Hilbert (1862 – 1943)

Ein neuer Gast kommt . . .

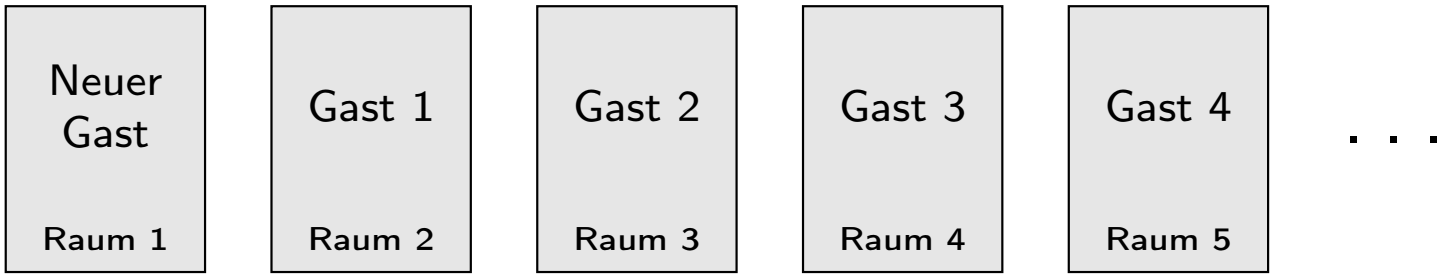


Unendlich viele neue Gäste kommen . . .

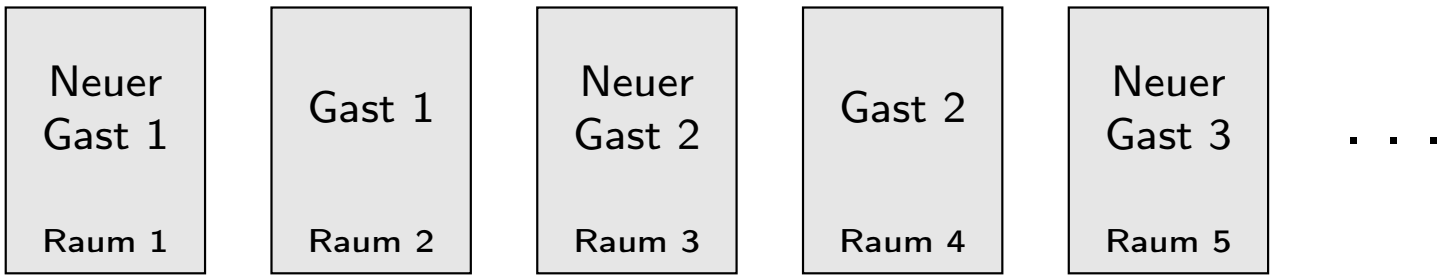


David Hilbert (1862 – 1943)

Ein neuer Gast kommt . . .



Unendlich viele neue Gäste kommen . . .



Überabzählbarkeit von $\mathbb{R}$

- Gibt es Mengen, die grösser sind als $\mathbb{N}$?
- Betrachten wir $\mathbb{R}$ (reelle Zahlen)
- Zur Erinnerung: $\pi, \sqrt{2} \in \mathbb{R}$, aber $\pi, \sqrt{2} \notin \mathbb{Q}$

Überabzählbarkeit von $\mathbb{R}$

- Gibt es Mengen, die grösser sind als $\mathbb{N}$?
- Betrachten wir $\mathbb{R}$ (reelle Zahlen)
- Zur Erinnerung: $\pi, \sqrt{2} \in \mathbb{R}$, aber $\pi, \sqrt{2} \notin \mathbb{Q}$

Satz

$\mathbb{R}$ *ist nicht abzählbar (wir sagen* ***überabzählbar****)*

Überabzählbarkeit von $\mathbb{R}$

- Gibt es Mengen, die grösser sind als $\mathbb{N}$?
- Betrachten wir $\mathbb{R}$ (reelle Zahlen)
- Zur Erinnerung: $\pi, \sqrt{2} \in \mathbb{R}$, aber $\pi, \sqrt{2} \notin \mathbb{Q}$

Satz

$\mathbb{R}$ *ist nicht abzählbar (wir sagen* ***überabzählbar****)*

- Wir führen wieder einen **Widerspruchsbeweis**
- Wie nehmen das Gegenteil an („$\mathbb{R}$ ist abzählbar“)
- Dann zeigen wir, dass daraus ein Widerspruch folgt

Überabzählbarkeit von $\mathbb{R}$

- Angenommen, $\mathbb{R}$ ist abzählbar
- Wir gucken sogar nur die reellen Zahlen zwischen 0 und 1 an
- Nehmen wir an, wir könnten diese Zahlen aufzählen

Überabzählbarkeit von $\mathbb{R}$

- Angenommen, $\mathbb{R}$ ist abzählbar
- Wir gucken sogar nur die reellen Zahlen zwischen 0 und 1 an
- Nehmen wir an, wir könnten diese Zahlen aufzählen

⇨ Dann erhalten wir eine Tabelle wie diese:

Nummer	Reelle Zahl
0	0. 2 5 6 5 1 4 0 5 ...
1	0. 6 8 0 0 7 1 4 3
2	0. 6 1 7 3 9 0 1 9
3	0. 8 8 7 4 0 8 4 8
⋮	⋮ ⋱

Überabzählbarkeit von $\mathbb{R}$

Nummer	Reelle Zahl									
0	0.	2	5	6	5	1	4	0	5	
1	0.	6	8	0	0	7	1	4	3	...
2	0.	6	1	7	3	9	0	1	9	
3	0.	8	8	7	4	0	8	4	8	
⋮				⋮						⋱

- Betrachte die i-te Nachkommastelle der i-ten Zahl

Überabzählbarkeit von $\mathbb{R}$

Nummer	Reelle Zahl									
0	0.	**2**	5	6	5	1	4	0	5	
1	0.	6	**8**	0	0	7	1	4	3	...
2	0.	6	1	**7**	3	9	0	1	9	
3	0.	8	8	7	**4**	0	8	4	8	
⋮				⋮						⋱

- Betrachte die i-te Nachkommastelle der i-ten Zahl
- Wir konstruieren eine reelle Zahl x
- x ist an der i-ten Stelle unterschiedlich von der i-ten Zahl
- Hier zum Beispiel $x = 0.1763\ldots$

Überabzählbarkeit von $\mathbb{R}$

Nummer	Reelle Zahl									
0	0.	**2**	5	6	5	1	4	0	5	
1	0.	6	**8**	0	0	7	1	4	3	...
2	0.	6	1	**7**	3	9	0	1	9	
3	0.	8	8	7	**4**	0	8	4	8	
⋮				⋮						⋱

- Betrachte die i-te Nachkommastelle der i-ten Zahl
- Wir konstruieren eine reelle Zahl x
- x ist an der i-ten Stelle unterschiedlich von der i-ten Zahl
- Hier zum Beispiel $x = 0.1763\ldots$

⇨ x kann nicht in der Tabelle stehen

Überabzählbarkeit von $\mathbb{R}$

Nummer	Reelle Zahl									
0	0.	**2**	5	6	5	1	4	0	5	
1	0.	6	**8**	0	0	7	1	4	3	...
2	0.	6	1	**7**	3	9	0	1	9	
3	0.	8	8	7	**4**	0	8	4	8	
⋮				⋮						⋱

- Betrachte die i-te Nachkommastelle der i-ten Zahl
- Wir konstruieren eine reelle Zahl x
- x ist an der i-ten Stelle unterschiedlich von der i-ten Zahl
- Hier zum Beispiel $x = 0.1763\ldots$

⇨ x kann nicht in der Tabelle stehen

Überabzählbarkeit von $\mathbb{R}$

Nummer	Reelle Zahl
0	0. **2** 5 6 5 1 4 0 5
1	0. 6 **8** 0 0 7 1 4 3 …
2	0. 6 1 **7** 3 9 0 1 9
3	0. 8 8 7 **4** 0 8 4 8
⋮	⋮ ⋱

- Betrachte die i-te Nachkommastelle der i-ten Zahl
- Wir konstruieren eine reelle Zahl x
- x ist an der i-ten Stelle unterschiedlich von der i-ten Zahl
- Hier zum Beispiel $x = 0.1763\ldots$

⇨ x kann nicht in der Tabelle stehen

⇨ Methode bekannt als **Diagonalisierung**

Teil 3 – Die Turing-Maschine

Teil 3 – Die Turing-Maschine

Die Turing-Maschine

Wir wollen jetzt zeigen, dass es Probleme gibt, die wir algorithmisch nicht lösen können; die Idee ist...

- Es gibt überabzählbar viele Probleme
- Es gibt abzählbar viele Algorithmen

Die Turing-Maschine

Wir wollen jetzt zeigen, dass es Probleme gibt, die wir algorithmisch nicht lösen können; die Idee ist...

- Es gibt überabzählbar viele Probleme
- Es gibt abzählbar viele Algorithmen

Aber zunächst müssen wir überlegen ...

- Was ist ein Problem?
- Was ist ein Algorithmus?

⇨ Wir brauchen eine saubere mathematische Definition

Was ist ein Problem?

- Ein **Alphabet** ist eine Menge von Symbolen
- Ein **Wort** ist eine Zeichenkette von diese Symbolen
- $DEZ = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist das Dezimalalphabet
- $x = 13817$ ist ein Wort „über“ DEZ

Was ist ein Problem?

- Ein **Alphabet** ist eine Menge von Symbolen
- Ein **Wort** ist eine Zeichenkette von diese Symbolen
- $DEZ = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist das Dezimalalphabet
- $x = 13817$ ist ein Wort „über“ DEZ

Definition

Ein **Entscheidungsproblem** L ist, für ein gegebenes Wort x, zu entscheiden, ob es in der Menge L ist

Was ist ein Problem?

- Ein **Alphabet** ist eine Menge von Symbolen
- Ein **Wort** ist eine Zeichenkette von diese Symbolen
- $DEZ = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist das Dezimalalphabet
- $x = 13817$ ist ein Wort „über“ DEZ

Definition

Ein **Entscheidungsproblem** L ist, für ein gegebenes Wort x, zu entscheiden, ob es in der Menge L ist

- Die Antwort ist immer JA oder NEIN
- Diese Modellierung ist tatsächlich sehr allgemein

Was ist ein Problem?

- $PRIM = \{y \mid y \text{ ist eine Primzahl}\}$ (y ist über DEZ)
- Ist x eine Primzahl?

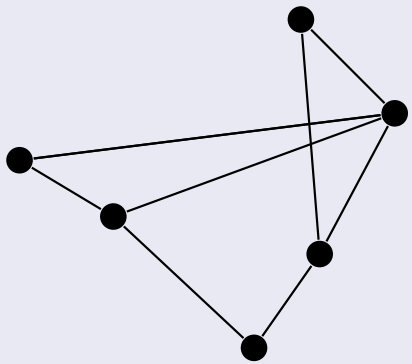
⇨ Ist $x \in PRIM$?

Was ist ein Problem?

- $PRIM = \{y \mid y \text{ ist eine Primzahl}\}$ (y ist über DEZ)
- Ist x eine Primzahl?

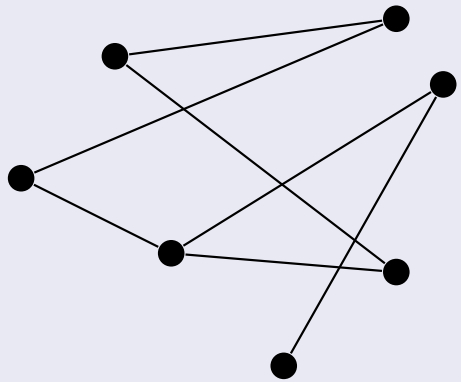
⇨ Ist $x \in PRIM$?

- $HK = \{y \mid y \text{ ist Graph mit einem Hamiltonkreis}\}$
- Der Graph



ist in HK

- Der Graph



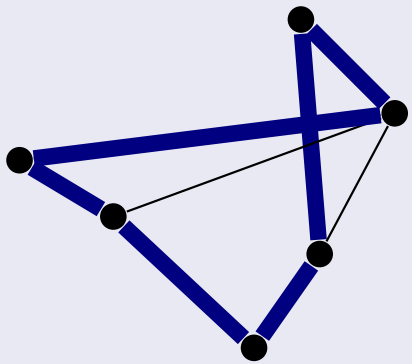
ist nicht in HK

Was ist ein Problem?

- $PRIM = \{y \mid y \text{ ist eine Primzahl}\}$ (y ist über DEZ)
- Ist x eine Primzahl?

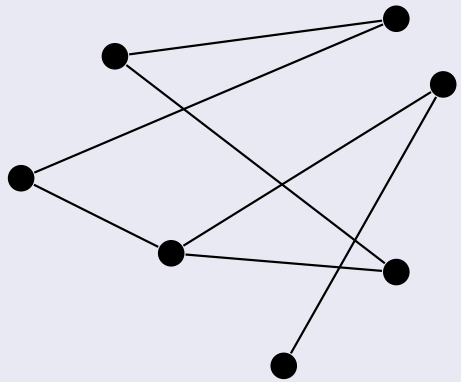
⇨ Ist $x \in PRIM$?

- $HK = \{y \mid y \text{ ist Graph mit einem Hamiltonkreis}\}$
- Der Graph



ist in HK

- Der Graph



ist nicht in HK

Was ist ein Algorithmus?

Wikimedia, Creative Commons

Definition (Wikipedia)

Ein **Algorithmus** ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten.

Was ist ein Algorithmus?

Wikimedia, Creative Commons

Definition (Wikipedia)

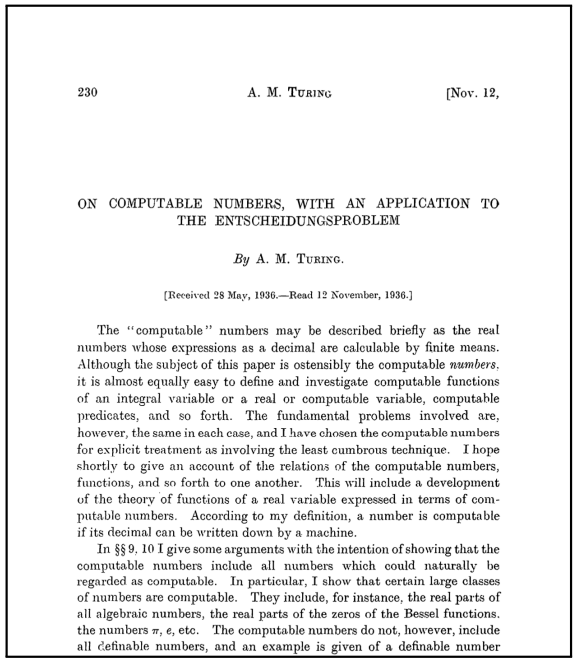
Ein **Algorithmus** ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten.

- Programm, das testet, ob eine Zahl eine Primzahl ist
- Rezept, Wegbeschreibung, Gebrauchsanweisung etc.

⇨ Kein mathematisches Objekt

Was ist ein Algorithmus?

Wikimedia, Creative Commons



230 A. M. TURING [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

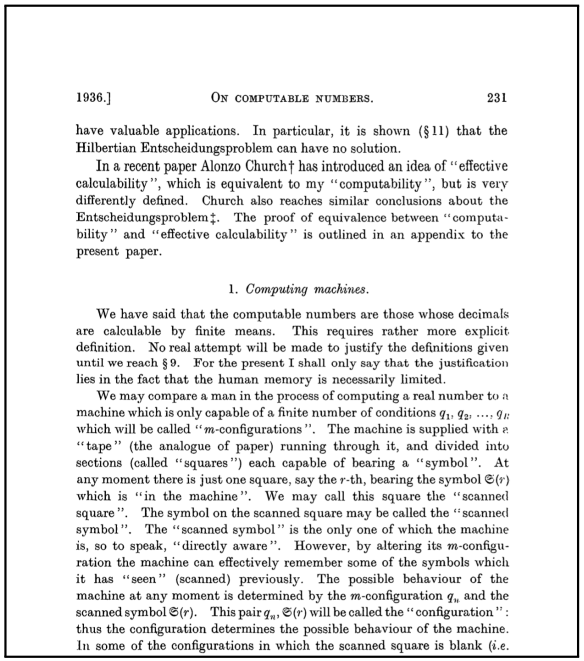
By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number

Oxford University Press



1936.] ON COMPUTABLE NUMBERS. 231

have valuable applications. In particular, it is shown (§ 11) that the Hilbertian Entscheidungsproblem can have no solution.

In a recent paper Alonzo Church† has introduced an idea of "effective calculability", which is equivalent to my "computability", but is very differently defined. Church also reaches similar conclusions about the Entscheidungsproblem‡. The proof of equivalence between "computability" and "effective calculability" is outlined in an appendix to the present paper.

1. *Computing machines.*

We have said that the computable numbers are those whose decimals are calculable by finite means. This requires rather more explicit definition. No real attempt will be made to justify the definitions given until we reach § 9. For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited.

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions $q_1, q_2, \ldots, q_R$ which will be called "m-configurations". The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its m-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair $q_n, \mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (*i.e.*

Oxford University Press

Was ist ein Algorithmus?

Ein Mathematiker hat Folgendes zur Verfügung

- einen Stift
- beliebig viel kariertes Papier
- eine endliche Anzahl von Regeln (Arithmetik, Logik, ...)

Was ist ein Algorithmus?

Ein Mathematiker hat Folgendes zur Verfügung

- einen Stift
- beliebig viel kariertes Papier
- eine endliche Anzahl von Regeln (Arithmetik, Logik, ...)

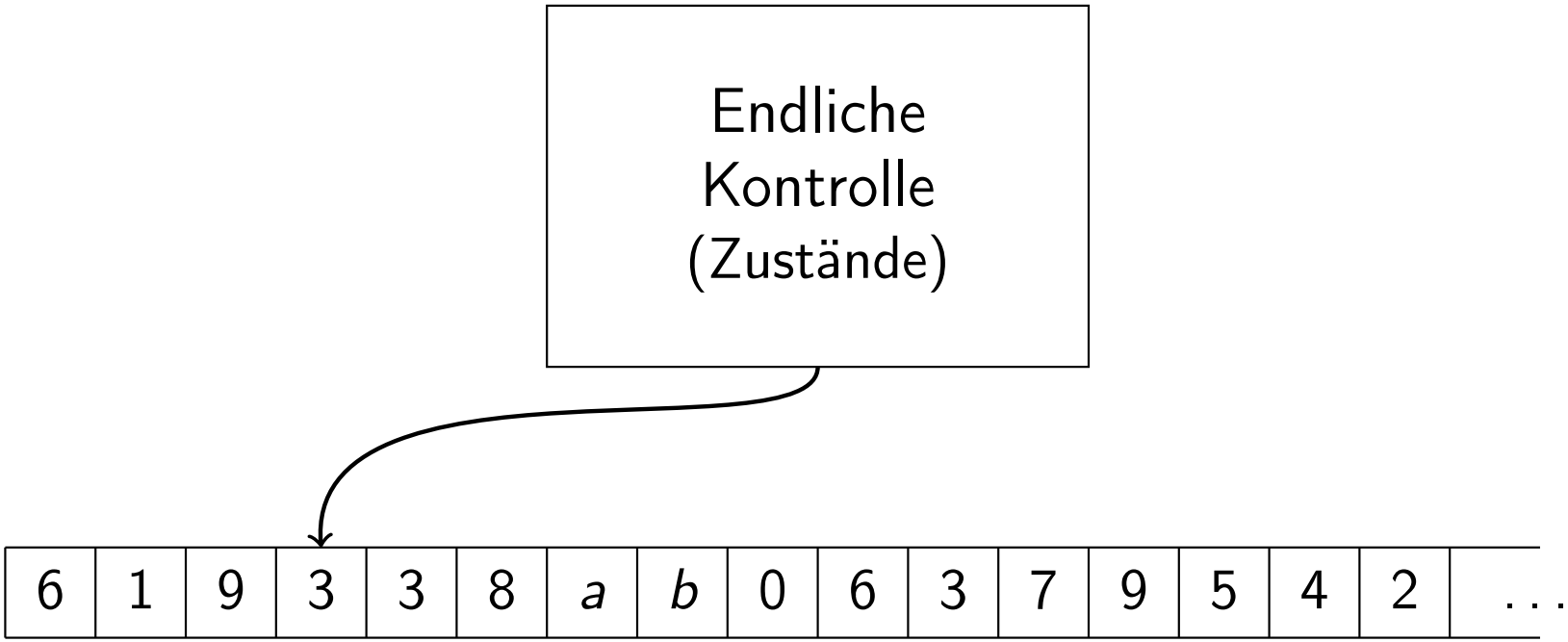
Eingabe: Zeichenkette (Wort) auf dem Blatt

Vorgehen: Der Mathematiker...

- betrachtet Zeichen, auf dem der Stift positioniert ist
- bewegt davon abhängig den Stift und schreibt
- beendet schliesslich seine Arbeit (er „hält“)

Ausgabe: Zeichenkette (Wort) auf dem Blatt

Turing-Maschine



Alonzo Church (1903 – 1995)

Wikimedia, Creative Commons

Church-Turing-These

Turing-Maschinen (TM) können genau das berechnen, was Algorithmen berechnen können

Alonzo Church (1903 – 1995)

Wikimedia, Creative Commons

Church-Turing-These

Turing-Maschinen (TM) können genau das berechnen, was Algorithmen berechnen können

Andersrum: Was eine TM nicht berechnen kann, kann auch kein Algorithmus berechnen (egal, ob Java, CPP oder Ruby on Rails)

Unentscheidbarkeit

- TM lösen Entscheidungsprobleme
- Ist $x \in PRIM$?
- $PRIM = \{y \mid y \text{ ist eine Primzahl}\}$

Unentscheidbarkeit

- TM lösen Entscheidungsprobleme
- Ist $x \in PRIM$?
- $PRIM = \{y \mid y \text{ ist eine Primzahl}\}$
- Sei M_{PRIM} eine TM für $PRIM$
- Ist $x \in PRIM$ „antwortet“ M_{PRIM} mit JA
- Ist $x \notin PRIM$ „antwortet“ M_{PRIM} mit NEIN

Unentscheidbarkeit

- TM lösen Entscheidungsprobleme
- Ist $x \in PRIM$?
- $PRIM = \{y \mid y \text{ ist eine Primzahl}\}$
- Sei M_{PRIM} eine TM für $PRIM$
- Ist $x \in PRIM$ „antwortet“ M_{PRIM} mit JA
- Ist $x \notin PRIM$ „antwortet“ M_{PRIM} mit NEIN

⇨ $PRIM = \{2, 3, 5, 7, 11, 13, 17, 19, \dots\}$

⇨ Entscheidungsprobleme sind unendliche Mengen von Wörtern

Unentscheidbarkeit

Satz (Turing)

*Es gibt Entscheidungsprobleme, die nicht von TM (Algorithmen) gelöst werden können (sie heissen **unentscheidbar**)*

- Zähle alle TM auf: $M_1, M_2, M_3, \ldots$
- Zähle alle Wörter (mit festem Alphabet) auf: $w_1, w_2, w_3, \ldots$
- Jede TM M_i löst ein Entscheidungsproblem
- Für jedes Wort w_j „antwortet“ M_i also mit JA oder NEIN

Unentscheidbarkeit

- Erzeuge wieder eine Tabelle
- Zeilen stehen für TM, Spalten für Wörter

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1										
M_2										
M_3										
M_4										
$\vdots$										

Unentscheidbarkeit

- Erzeuge wieder eine Tabelle
- Zeilen stehen für TM, Spalten für Wörter

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	0	1	1	1	1	0	0	
M_2	0	0	1	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- 1 in Zelle (i, j) wenn M_i für w_j JA antwortet
- 0 in Zelle (i, j) wenn M_i für w_j NEIN antwortet

Unentscheidbarkeit

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	1	1	1	1	1	0	0	
M_2	0	0	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

Unentscheidbarkeit

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	1	1	1	1	1	0	0	
M_2	0	0	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 antwortet JA für w_1

Unentscheidbarkeit

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	1	1	1	1	1	0	0	
M_2	0	0	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 antwortet JA für w_1
- M_2 antwortet NEIN für w_2

Unentscheidbarkeit

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	1	1	1	1	1	0	0	
M_2	0	0	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 antwortet JA für w_1
- M_2 antwortet NEIN für w_2
- M_3 antwortet NEIN für w_3

Unentscheidbarkeit

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	1	1	1	1	1	0	0	
M_2	0	0	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 antwortet JA für w_1
- M_2 antwortet NEIN für w_2
- M_3 antwortet NEIN für w_3
- M_4 antwortet JA für w_4

Unentscheidbarkeit

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	1	1	1	1	1	0	0	
M_2	0	0	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 antwortet JA für w_1
- M_2 antwortet NEIN für w_2
- M_3 antwortet NEIN für w_3
- M_4 antwortet JA für w_4
- Definiere Entscheidungsproblem *DIAG* (**Diagonalisierung**)
- w_i ist in *DIAG*, wenn M_i NEIN für w_i antwortet
- Hier $DIAG = \{w_2, w_3, \dots\}$

Unentscheidbarkeit

w_i ist in $DIAG$, wenn M_i NEIN für w_i antwortet

- Nehmen wir an, es gibt eine TM M für $DIAG$

⇨ M muss in der Aufzählung aller TM vorkommen

⇨ Sei M die k-te TM, also M_k

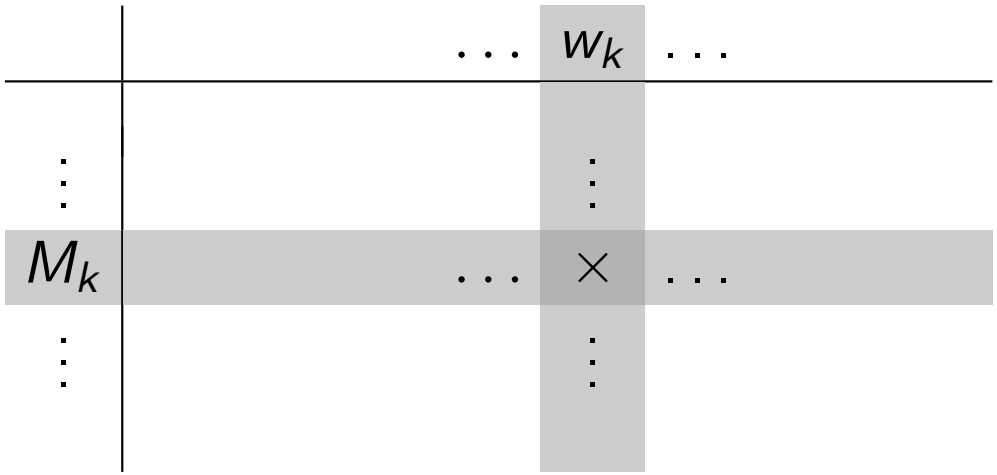
Unentscheidbarkeit

w_i ist in *DIAG*, wenn M_i NEIN für w_i antwortet

- Nehmen wir an, es gibt eine TM M für *DIAG*

⇨ M muss in der Aufzählung aller TM vorkommen

⇨ Sei M die k-te TM, also M_k; betrachte Zelle (k, k)



- Steht dort eine 1 oder eine 0?

Unentscheidbarkeit

w_i ist in $DIAG$, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet

Unentscheidbarkeit

w_i ist in $DIAG$, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in $DIAG$

Unentscheidbarkeit

w_i ist in *DIAG*, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in *DIAG*
- Aber M_k ist eine TM für *DIAG*

Unentscheidbarkeit

w_i ist in *DIAG*, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in *DIAG*
  - Aber M_k ist eine TM für *DIAG*
- ⇨ Also antwortet M_k für w_k mit NEIN

Unentscheidbarkeit

w_i ist in *DIAG*, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in *DIAG*
  - Aber M_k ist eine TM für *DIAG*
- ⇨ Also antwortet M_k für w_k mit NEIN

Unentscheidbarkeit

w_i ist in $DIAG$, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in $DIAG$
  - Aber M_k ist eine TM für $DIAG$
- ⇨ Also antwortet M_k für w_k mit NEIN

Also gut, dann steht in Zelle (k, k) eine 0

- ⇨ Dies bedeutet, dass M_k für w_k mit NEIN antwortet

Unentscheidbarkeit

w_i ist in *DIAG*, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in *DIAG*
  - Aber M_k ist eine TM für *DIAG*
- ⇨ Also antwortet M_k für w_k mit NEIN

Also gut, dann steht in Zelle (k, k) eine 0

- ⇨ Dies bedeutet, dass M_k für w_k mit NEIN antwortet
- ⇨ w_k ist nach Konstruktion in *DIAG*

Unentscheidbarkeit

w_i ist in $DIAG$, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in $DIAG$
  - Aber M_k ist eine TM für $DIAG$
- ⇨ Also antwortet M_k für w_k mit NEIN

Also gut, dann steht in Zelle (k, k) eine 0

- ⇨ Dies bedeutet, dass M_k für w_k mit NEIN antwortet
- ⇨ w_k ist nach Konstruktion in $DIAG$
  - Aber M_k ist eine TM für $DIAG$

Unentscheidbarkeit

w_i ist in $DIAG$, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in $DIAG$
  - Aber M_k ist eine TM für $DIAG$
- ⇨ Also antwortet M_k für w_k mit NEIN

Also gut, dann steht in Zelle (k, k) eine 0

- ⇨ Dies bedeutet, dass M_k für w_k mit NEIN antwortet
- ⇨ w_k ist nach Konstruktion in $DIAG$
  - Aber M_k ist eine TM für $DIAG$
- ⇨ Also antwortet M_k für w_k mit JA

Unentscheidbarkeit

w_i ist in $DIAG$, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in $DIAG$
  - Aber M_k ist eine TM für $DIAG$
- ⇨ Also antwortet M_k für w_k mit NEIN

Also gut, dann steht in Zelle (k, k) eine 0

- ⇨ Dies bedeutet, dass M_k für w_k mit NEIN antwortet
- ⇨ w_k ist nach Konstruktion in $DIAG$
  - Aber M_k ist eine TM für $DIAG$
- ⇨ Also antwortet M_k für w_k mit JA

Unentscheidbarkeit

w_i ist in $DIAG$, wenn M_i NEIN für w_i antwortet

Angenommen, in Zelle (k, k) steht eine 1

- ⇨ Dies bedeutet, dass M_k für w_k mit JA antwortet
- ⇨ w_k ist nach Konstruktion nicht in $DIAG$
  - Aber M_k ist eine TM für $DIAG$
- ⇨ Also antwortet M_k für w_k mit NEIN

Also gut, dann steht in Zelle (k, k) eine 0

- ⇨ Dies bedeutet, dass M_k für w_k mit NEIN antwortet
- ⇨ w_k ist nach Konstruktion in $DIAG$
  - Aber M_k ist eine TM für $DIAG$
- ⇨ Also antwortet M_k für w_k mit JA

Das Halteproblem

```
print "Hello, world.";
exit(0);
```

Das Halteproblem

```
print "Hello, world.";
exit(0);
```

```
while ( true ) { }
print "Hello, world.";
exit(0);
```

Das Halteproblem

```
print "Hello, world.";
exit(0);
```

```
while ( true ) { }
print "Hello, world.";
exit(0);
```

```
$n=3;
$total=3;

while (true) {
   for ($x=1; $x <= $total-2; $x++) {
      for ($y=1; $y <= $total-$x-1; $y++) {
         $z = $total-$x-$y;
         if ($x**$n+$y**$n == $z**$n) {
            print "Hello, world";
            exit(0);
         }
      }
   }
   $total=$total+1;
}
```

Das Halteproblem

```
print "Hello, world.";
exit(0);
```

```
while ( true ) { }
print "Hello, world.";
exit(0);
```

```
$n=3;        # Dieses Programm terminiert, wenn x, y und z existieren,
$total=3;    # so dass x^n+y^n=z^n, wobei n mindestens 3 ist
             # Fermats letzter Satz, offen für 300 Jahre bis 1994
while (true) {
   for ($x=1; $x <= $total-2; $x++) {
      for ($y=1; $y <= $total-$x-1; $y++) {
         $z = $total-$x-$y;
         if ($x**$n+$y**$n == $z**$n) {
            print "Hello, world";
            exit(0);
         }
      }
   }
   $total=$total+1;
}
```

Teil 4 – Schnelle Turing-Maschinen

Teil 4 – Schnelle Turing-Maschinen

Laufzeit-Analyse

- Betrachte nun Entscheidungsprobleme, die von TM gelöst werden können

⇨ Zum Beispiel *PRIM*

Laufzeit-Analyse

- Betrachte nun Entscheidungsprobleme, die von TM gelöst werden können

⇨ Zum Beispiel *PRIM*

- Wie schnell geht dies?
- Uns interessiert die Anzahl Berechnungsschritte

Laufzeit-Analyse

- Betrachte nun Entscheidungsprobleme, die von TM gelöst werden können

⇨ Zum Beispiel *PRIM*

- Wie schnell geht dies?
- Uns interessiert die Anzahl Berechnungsschritte
- $x \in PRIM$ kann sicher schneller für $x = 5$ als für $x = 10\,000\,000\,013$ entschieden werden
- Generell steigt die „Laufzeit“ wenn die Eingabelänge n steigt
- Es gilt ungefähr: $n = \log_2 x$

Laufzeit-Analyse

- Betrachte nun Entscheidungsprobleme, die von TM gelöst werden können

⇨ Zum Beispiel *PRIM*

- Wie schnell geht dies?
- Uns interessiert die Anzahl Berechnungsschritte
- $x \in PRIM$ kann sicher schneller für $x = 5$ als für $x = 10\,000\,000\,013$ entschieden werden
- Generell steigt die „Laufzeit“ wenn die Eingabelänge n steigt
- Es gilt ungefähr: $n = \log_2 x$

⇨ Aber wie schnell wächst die Laufzeit?

Laufzeit-Analyse

n	10	50	100	300	10 000
$10n$	100	500	1 000	3 000	100 000
$4n^2$	400	10 000	40 000	360 000	400 000 000
n^3	1 000	125 000	1 000 000	27 000 000	13 Ziffern
2^n	1024	16 Ziffern	31 Ziffern	91 Ziffern	3 011 Ziffern
3^n	59 049	24 Ziffern	48 Ziffern	143 Ziffern	4 772 Ziffern

Laufzeit-Analyse

n	10	50	100	300	10 000
$10n$	100	500	1 000	3 000	100 000
$4n^2$	400	10 000	40 000	360 000	400 000 000
n^3	1 000	125 000	1 000 000	27 000 000	13 Ziffern
2^n	1024	16 Ziffern	31 Ziffern	91 Ziffern	3 011 Ziffern
3^n	59 049	24 Ziffern	48 Ziffern	143 Ziffern	4 772 Ziffern

Definition

Eine Funktion f ist in $\mathcal{O}(g)$ für eine Funktion g, wenn (für grosse Werte von n) f nur konstant schneller wächst als g

Laufzeit-Analyse

n	10	50	100	300	10 000
$10n$	100	500	1 000	3 000	100 000
$4n^2$	400	10 000	40 000	360 000	400 000 000
n^3	1 000	125 000	1 000 000	27 000 000	13 Ziffern
2^n	1024	16 Ziffern	31 Ziffern	91 Ziffern	3 011 Ziffern
3^n	59 049	24 Ziffern	48 Ziffern	143 Ziffern	4 772 Ziffern

Definition

Eine Funktion f ist in $\mathcal{O}(g)$ für eine Funktion g, wenn (für grosse Werte von n) f nur konstant schneller wächst als g

- $4n^2 \in \mathcal{O}(n^2)$
- $100n^3 \in \mathcal{O}(n^3)$
- $100n^3 + 50n \in \mathcal{O}(n^3)$
- $n^3 \notin \mathcal{O}(n^2)$
- $5n \in \mathcal{O}(n^2)$
- $2^n \notin \mathcal{O}(n^2)$

Laufzeit-Analyse

Betrachten wir einen einfachen Primzahltest

```
$zahl=$ARGV[0];
$test=2;

while ($test < $zahl) {
   $rest=$zahl%$test;
   if ($rest == 0) {
      print "NEIN";
      exit(0);
   }
   $test=$test+1;
}
print "JA";
exit(0);
```

Laufzeit-Analyse

Betrachten wir einen einfachen Primzahltest

```
$zahl=$ARGV[0];
$test=2;

while ($test < $zahl) {
   $rest=$zahl%$test;
   if ($rest == 0) {
      print "NEIN";
      exit(0);
   }
   $test=$test+1;
}
print "JA";
exit(0);
```

Wenn `zahl` prim, wird die Schleife $\approx$ `$zahl` $\approx 2^n$ Mal durchlaufen

Laufzeit-Analyse

Betrachten wir einen einfachen Primzahltest

```
$zahl=$ARGV[0];
$test=2;

while ($test < $zahl) {
   $rest=$zahl%$test;
   if ($rest == 0) {
      print "NEIN";
      exit(0);
   }
   $test=$test+1;
}
print "JA";
exit(0);
```

```
$zahl=$ARGV[0];
$test=2;
$wurzel_zahl=sqrt($zahl);

while ($test <= $wurzel_zahl) {
  $rest=$zahl%$test;
   if ($rest == 0) {
      print "NEIN";
      exit(0);
   }
  $test=$test+1;
}
print "JA";
exit(0);
```

Wenn `zahl` prim, wird die Schleife $\approx$ `$zahl` $\approx 2^n$ Mal durchlaufen

Laufzeit-Analyse

Betrachten wir einen einfachen Primzahltest

```
$zahl=$ARGV[0];
$test=2;

while ($test < $zahl) {
   $rest=$zahl%$test;
   if ($rest == 0) {
      print "NEIN";
      exit(0);
   }
   $test=$test+1;
}
print "JA";
exit(0);
```

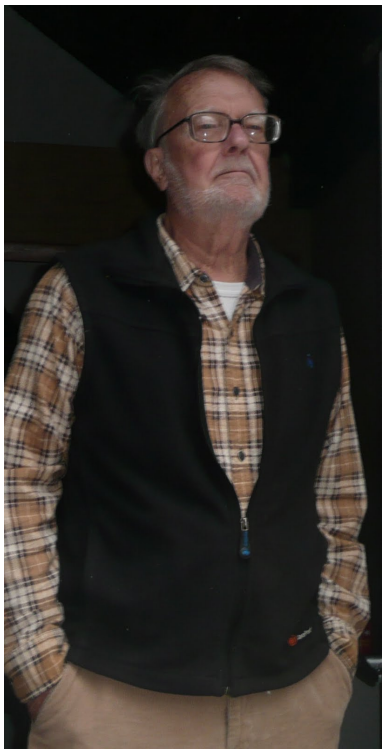
Wenn `zahl` prim, wird die Schleife $\approx$ `$zahl` $\approx 2^n$ Mal durchlaufen

```
$zahl=$ARGV[0];
$test=2;
$wurzel_zahl=sqrt($zahl);

while ($test <= $wurzel_zahl) {
  $rest=$zahl%$test;
   if ($rest == 0) {
      print "NEIN";
      exit(0);
   }
  $test=$test+1;
}
print "JA";
exit(0);
```

Es reicht bis zur Wurzel von `$zahl` zu testen; Schleife wird $\approx \sqrt{\texttt{\$zahl}} \approx 1.41^n$ Mal durchlaufen

Alan Cobham (*1927), Jack Edmonds (*1934)



Unbekannter Urheber

Unbekannter Urheber

These von C. und E.

Effiziente Algorithmen sind solche, die in polynomieller Zeit laufen

Die Laufzeit ist in $\mathcal{O}(n^k)$ für ein $k \in \mathbb{N}$

Effiziente Algorithmen

- Der Begriff ist unabhängig vom jeweiligen Berechnungsmodell

⇨ Wenn ein „echter Algorithmus" etwas in $\mathcal{O}(n^k)$ berechnen kann, kann eine TM es in $\mathcal{O}(n^{k'})$

- Wir nennen die Klasse der Polynome deswegen **robust**

Effiziente Algorithmen

- Der Begriff ist unabhängig vom jeweiligen Berechnungsmodell

⇨ Wenn ein „echter Algorithmus" etwas in $\mathcal{O}(n^k)$ berechnen kann, kann eine TM es in $\mathcal{O}(n^{k'})$

- Wir nennen die Klasse der Polynome deswegen **robust**

Definition

Die Klasse $\mathcal{P}$ enthält alle Entscheidungsprobleme, die effizient von TM gelöst werden können

Effiziente Algorithmen

- Wir interessieren uns für die Laufzeit im **schlechtesten Fall**

⇨ Für unseren Primzahltest ist dies, wenn die Eingabe prim ist

⇨ Wäre sie gerade, würden beide Algorithmen schnell sein

Effiziente Algorithmen

- Wir interessieren uns für die Laufzeit im **schlechtesten Fall**

⇨ Für unseren Primzahltest ist dies, wenn die Eingabe prim ist

⇨ Wäre sie gerade, würden beide Algorithmen schnell sein

- Unsere Primzahl-Algorithmen sind beide nicht effizient

Effiziente Algorithmen

- Wir interessieren uns für die Laufzeit im **schlechtesten Fall**

⇨ Für unseren Primzahltest ist dies, wenn die Eingabe prim ist

⇨ Wäre sie gerade, würden beide Algorithmen schnell sein

- Unsere Primzahl-Algorithmen sind beide nicht effizient
- Allerdings existiert ein solcher Algorithmus

⇨ $PRIM \in \mathcal{P}$

Effiziente Algorithmen

- Wir interessieren uns für die Laufzeit im **schlechtesten Fall**

⇨ Für unseren Primzahltest ist dies, wenn die Eingabe prim ist

⇨ Wäre sie gerade, würden beide Algorithmen schnell sein

- Unsere Primzahl-Algorithmen sind beide nicht effizient
- Allerdings existiert ein solcher Algorithmus

⇨ $PRIM \in \mathcal{P}$

- Was ist mit Entscheidungsproblemen, die nicht in $\mathcal{P}$ liegen?

Teil 5 – $\mathcal{NP}$-Vollständigkeit

Teil 5 – $\mathcal{NP}$-Vollständigkeit

Polynomzeit-Verifizierer

- Betrachte alternatives Modell
- Ein **Polynomzeit-Verifizierer** (PV) muss die Lösung eines Problems nicht selber berechnen

Polynomzeit-Verifizierer

- Betrachte alternatives Modell
- Ein **Polynomzeit-Verifizierer** (PV) muss die Lösung eines Problems nicht selber berechnen
- Er erhält mit der Eingabe einen „(potentiellen) Zeugen“
- Der Zeuge ist eine Zeichenkette

Polynomzeit-Verifizierer

- Betrachte alternatives Modell
- Ein **Polynomzeit-Verifizierer** (PV) muss die Lösung eines Problems nicht selber berechnen
- Er erhält mit der Eingabe einen „(potentiellen) Zeugen“
- Der Zeuge ist eine Zeichenkette

⇨ Ist Eingabe „JA-Eingabe“ beweist der Zeuge dies

⇨ Ist Eingabe „NEIN-Eingabe“ ist der Zeuge beliebig

Polynomzeit-Verifizierer

- Betrachte alternatives Modell
- Ein **Polynomzeit-Verifizierer** (PV) muss die Lösung eines Problems nicht selber berechnen
- Er erhält mit der Eingabe einen „(potentiellen) Zeugen“
- Der Zeuge ist eine Zeichenkette

⇨ Ist Eingabe „JA-Eingabe“ beweist der Zeuge dies

⇨ Ist Eingabe „NEIN-Eingabe“ ist der Zeuge beliebig

⇨ PV muss Eingabe und Zeugen **verifizieren**

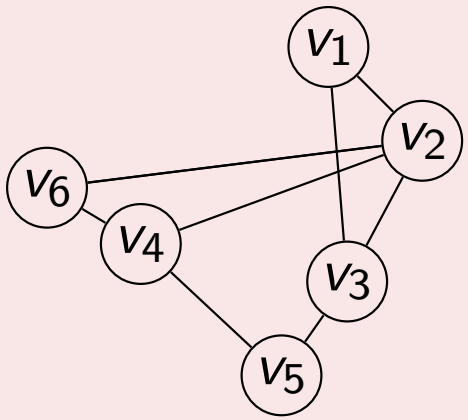
Hamiltonkreisproblem HK

$$HK = \{y \mid y \text{ ist Graph mit einem Hamiltonkreis}\}$$

Hamiltonkreisproblem HK

$HK = \{y \mid y \text{ ist Graph mit einem Hamiltonkreis}\}$

TM erhält Eingabe

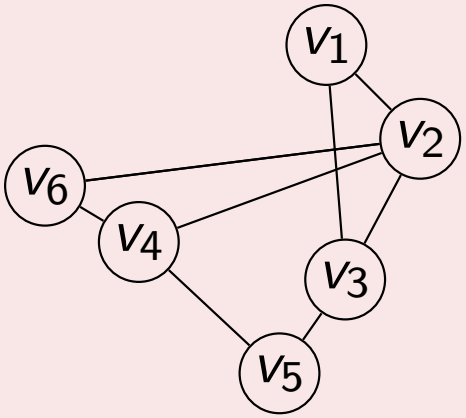


und muss Ausgabe JA oder NEIN berechnen

Hamiltonkreisproblem HK

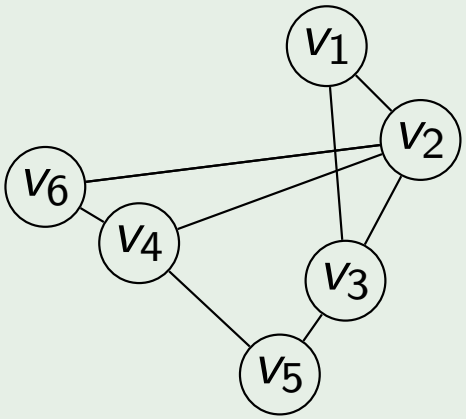
$HK = \{y \mid y \text{ ist Graph mit einem Hamiltonkreis}\}$

TM erhält Eingabe



und muss Ausgabe JA oder NEIN berechnen

PV erhält Eingabe



und Zeugen

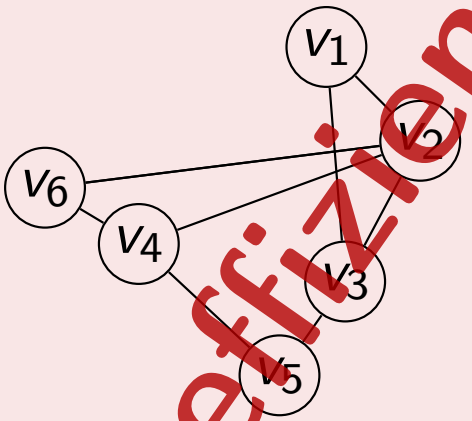
$$v_1, v_3, v_5, v_4, v_6, v_2, v_1$$

und muss diesen mit JA oder NEIN verifizieren

Hamiltonkreisproblem HK

$HK = \{y \mid y \text{ ist Graph mit einem Hamiltonkreis}\}$

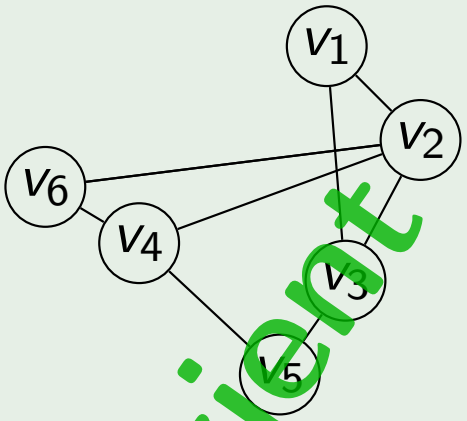
TM erhält Eingabe



und muss Ausgabe JA oder NEIN berechnen

nicht effizient

PV erhält Eingabe



und Zeugen

$v_1, v_3, v_5, v_4, v_6, v_2, v_1$

effizient

und muss diesen mit JA oder NEIN verifizieren

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

Die Klasse $\mathcal{NP}$ enthält alle Entscheidungsprobleme, die effizient von PV gelöst werden können

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

Die Klasse $\mathcal{NP}$ enthält alle Entscheidungsprobleme, die effizient von PV gelöst werden können

- „$\mathcal{N}$“ steht für „nichtdeterministisch“

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

Die Klasse $\mathcal{NP}$ enthält alle Entscheidungsprobleme, die effizient von PV gelöst werden können

- „$\mathcal{N}$“ steht für „nichtdeterministisch“
- PV haben gegenüber TM einen offensichtlichen Vorteil
- Wenn wir etwas mit einer TM effizient berechnen können, dann auch mit einer PV

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

Die Klasse $\mathcal{NP}$ enthält alle Entscheidungsprobleme, die effizient von PV gelöst werden können

- „$\mathcal{N}$“ steht für „nichtdeterministisch“
- PV haben gegenüber TM einen offensichtlichen Vorteil
- Wenn wir etwas mit einer TM effizient berechnen können, dann auch mit einer PV

$\Rightarrow \mathcal{P} \subseteq \mathcal{NP}$

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

Die Klasse $\mathcal{NP}$ enthält alle Entscheidungsprobleme, die effizient von PV gelöst werden können

- „$\mathcal{N}$“ steht für „nichtdeterministisch“
- PV haben gegenüber TM einen offensichtlichen Vorteil
- Wenn wir etwas mit einer TM effizient berechnen können, dann auch mit einer PV

⇨ $\mathcal{P} \subseteq \mathcal{NP}$

- Aber was ist mit der anderen Richtung?

⇨ Wir wissen bis heute nicht, ob $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{P} \subsetneq \mathcal{NP}$

$\mathcal{P}$ versus $\mathcal{NP}$

- Es wird vermutet, dass $\mathcal{P} \subsetneq \mathcal{NP}$

⇨ Es gibt vermutlich Probleme in $\mathcal{NP}$, für die es keine effizienten TM gibt

$\mathcal{P}$ versus $\mathcal{NP}$

- Es wird vermutet, dass $\mathcal{P} \subsetneq \mathcal{NP}$

⇨ Es gibt vermutlich Probleme in $\mathcal{NP}$, für die es keine effizienten TM gibt

- Seit Jahrzehnten ist es offen, ein solches zu finden

$\mathcal{P}$ versus $\mathcal{NP}$

- Es wird vermutet, dass $\mathcal{P} \subsetneq \mathcal{NP}$

⇨ Es gibt vermutlich Probleme in $\mathcal{NP}$, für die es keine effizienten TM gibt

- Seit Jahrzehnten ist es offen, ein solches zu finden
- Was können wir tun?

⇨ Identifiziere „schwerste Probleme“ in $\mathcal{NP}$

$\mathcal{P}$ versus $\mathcal{NP}$

- Es wird vermutet, dass $\mathcal{P} \subsetneq \mathcal{NP}$

⇨ Es gibt vermutlich Probleme in $\mathcal{NP}$, für die es keine effizienten TM gibt

- Seit Jahrzehnten ist es offen, ein solches zu finden
- Was können wir tun?

⇨ Identifiziere „schwerste Probleme“ in $\mathcal{NP}$

Definition

Ein Problem A in $\mathcal{NP}$ wird als **$\mathcal{NP}$-vollständig** bezeichnet, wenn die effiziente Lösbarkeit von A die effiziente Lösbarkeit aller Probleme in $\mathcal{NP}$ erlaubt

Stephen A. Cook (*1939)

Wikimedia, Creative Commons

Satz von Cook

Es existiert ein $\mathcal{NP}$-vollständiges Entscheidungsproblem

Richard M. Karp (*1935)

Wikimedia, Creative Commons

Karps 21 Probleme

Es existieren 21 weitere $\mathcal{NP}$-vollständige Entscheidungsprobleme

Richard M. Karp (*1935)

Wikimedia, Creative Commons

Karps 21 Probleme

Es existieren 21 weitere $\mathcal{NP}$-vollständige Entscheidungsprobleme

Heute kennen wir Tausende, und doch können wir für keines zeigen, dass es keine effiziente TM gibt

Polynomzeit-Reduktionen

Voraussetzung

- Seien A und B zwei Probleme in $\mathcal{NP}$

Polynomzeit-Reduktionen

Voraussetzung

- Seien A und B zwei Probleme in $\mathcal{NP}$
- Für A wissen wir bereits, dass es $\mathcal{NP}$-vollständig ist

⇨ Wenn A effizient gelöst werden kann, dann auch alle anderen

Polynomzeit-Reduktionen

Voraussetzung

- Seien A und B zwei Probleme in $\mathcal{NP}$
- Für A wissen wir bereits, dass es $\mathcal{NP}$-vollständig ist

⇨ Wenn A effizient gelöst werden kann, dann auch alle anderen

Vorgehen

- Zeige: Wenn B effizient gelöst werden kann, dann auch A

Polynomzeit-Reduktionen

Voraussetzung

- Seien A und B zwei Probleme in $\mathcal{NP}$
- Für A wissen wir bereits, dass es $\mathcal{NP}$-vollständig ist

⇨ Wenn A effizient gelöst werden kann, dann auch alle anderen

Vorgehen

- Zeige: Wenn B effizient gelöst werden kann, dann auch A

⇨ Wenn B effizient gelöst werden kann, dann auch alle anderen

Polynomzeit-Reduktionen

Voraussetzung

- Seien A und B zwei Probleme in $\mathcal{NP}$
- Für A wissen wir bereits, dass es $\mathcal{NP}$-vollständig ist

⇨ Wenn A effizient gelöst werden kann, dann auch alle anderen

Vorgehen

- Zeige: Wenn B effizient gelöst werden kann, dann auch A

⇨ Wenn B effizient gelöst werden kann, dann auch alle anderen

⇨ B ist $\mathcal{NP}$-vollständig

Polynomzeit-Reduktionen

Voraussetzung

- Seien A und B zwei Probleme in $\mathcal{NP}$
- Für A wissen wir bereits, dass es $\mathcal{NP}$-vollständig ist

⇨ Wenn A effizient gelöst werden kann, dann auch alle anderen

Vorgehen

- Zeige: Wenn B effizient gelöst werden kann, dann auch A

⇨ Wenn B effizient gelöst werden kann, dann auch alle anderen

⇨ B ist $\mathcal{NP}$-vollständig

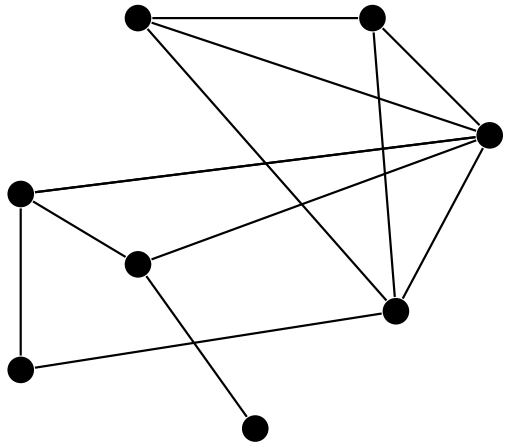
- Wir „reduzieren“ das Lösen von A auf das Lösen von B

⇨ Dies wird als **Polynomzeit-Reduktion** bezeichnet

Polynomzeit-Reduktionen

Definition

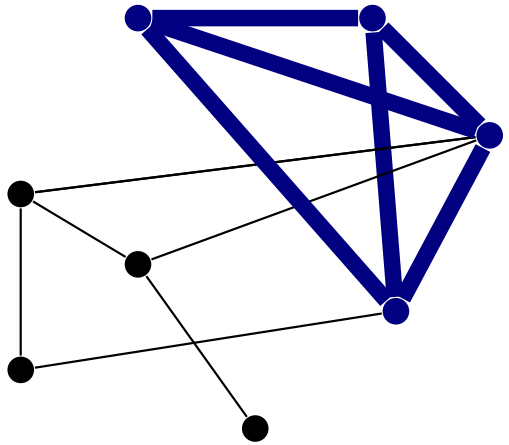
Eine Clique in einem Graphen ist eine Menge von Knoten, die alle untereinander verbunden sind



Polynomzeit-Reduktionen

Definition

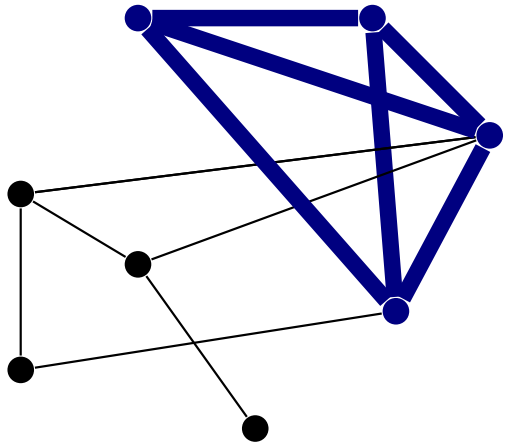
Eine Clique in einem Graphen ist eine Menge von Knoten, die alle untereinander verbunden sind



Polynomzeit-Reduktionen

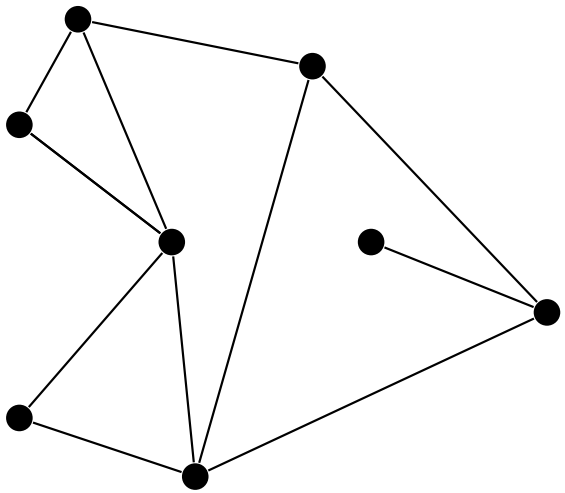
Definition

Eine Clique in einem Graphen ist eine Menge von Knoten, die alle untereinander verbunden sind



Definition

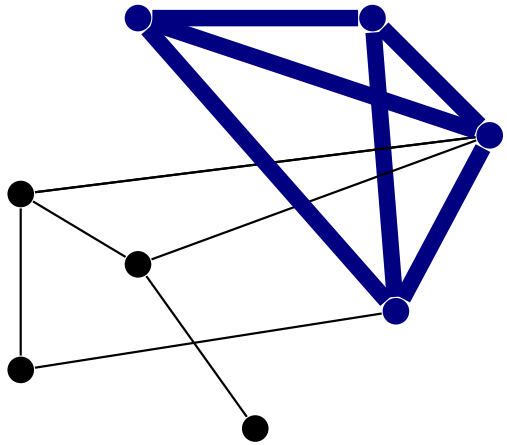
Eine Independent-Set (IS) in einem Graphen ist eine Menge von Knoten, von denen keine miteinander verbunden sind



Polynomzeit-Reduktionen

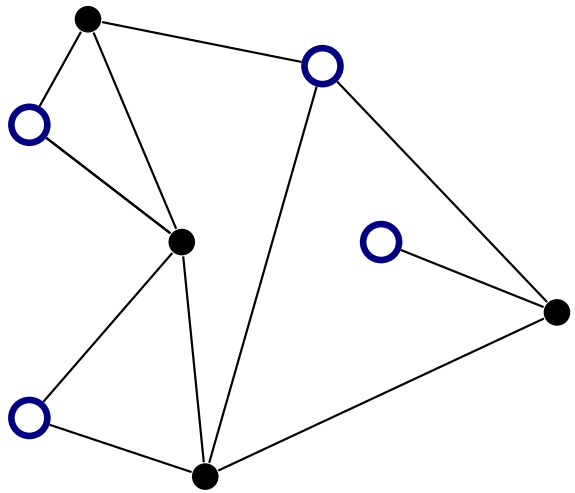
Definition

Eine Clique in einem Graphen ist eine Menge von Knoten, die alle untereinander verbunden sind



Definition

Eine Independent-Set (IS) in einem Graphen ist eine Menge von Knoten, von denen keine miteinander verbunden sind



Polynomzeit-Reduktionen

Betrachte folgende Entscheidungsprobleme

- *CLIQUE*

  $= \{(y, k) \mid y \text{ ist Graph mit Clique der Grösse } k\}$

- *IND-SET*

  $= \{(y, k) \mid y \text{ ist Graph mit IS der Grösse } k\}$

Polynomzeit-Reduktionen

Betrachte folgende Entscheidungsprobleme

- *CLIQUE*

  $= \{(y, k) \mid y \text{ ist Graph mit Clique der Grösse } k\}$
- *IND-SET*

  $= \{(y, k) \mid y \text{ ist Graph mit IS der Grösse } k\}$

Vorgehen

- Für *CLIQUE* wissen wir, dass es $\mathcal{NP}$-vollständig ist

⇨ Zeige, dass *IND-SET* dann auch $\mathcal{NP}$-vollständig ist

Polynomzeit-Reduktionen

Betrachte folgende Entscheidungsprobleme

- *CLIQUE*

  $= \{(y, k) \mid y \text{ ist Graph mit Clique der Grösse } k\}$
- *IND-SET*

  $= \{(y, k) \mid y \text{ ist Graph mit IS der Grösse } k\}$

Vorgehen

- Für *CLIQUE* wissen wir, dass es $\mathcal{NP}$-vollständig ist

⇨ Zeige, dass *IND-SET* dann auch $\mathcal{NP}$-vollständig ist

⇨ „Reduziere *CLIQUE* auf *IND-SET*“

Polynomzeit-Reduktionen

Betrachte folgende Entscheidungsprobleme

- *CLIQUE*

  $= \{(y, k) \mid y \text{ ist Graph mit Clique der Grösse } k\}$
- *IND-SET*

  $= \{(y, k) \mid y \text{ ist Graph mit IS der Grösse } k\}$

Vorgehen

- Für *CLIQUE* wissen wir, dass es $\mathcal{NP}$-vollständig ist

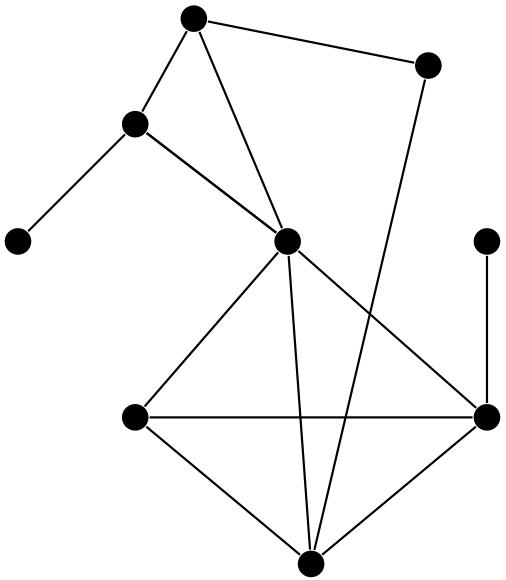
⇨ Zeige, dass *IND-SET* dann auch $\mathcal{NP}$-vollständig ist

⇨ „Reduziere *CLIQUE* auf *IND-SET*“

⇨ Könnten wir *IND-SET* effizient lösen, dann auch *CLIQUE*

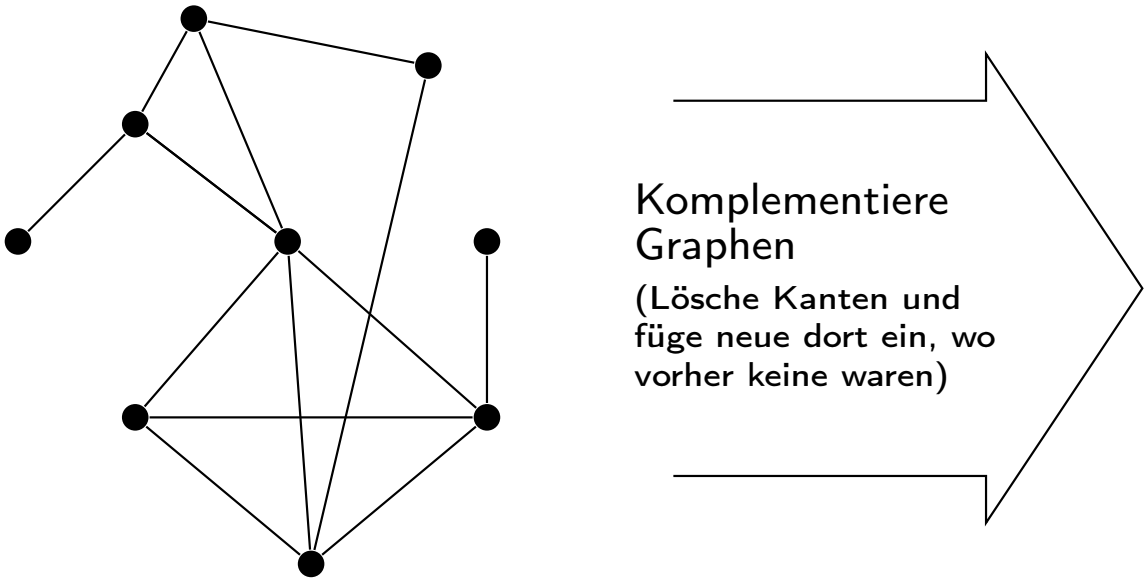
Polynomzeit-Reduktionen

Wir machen folgende Beobachtung



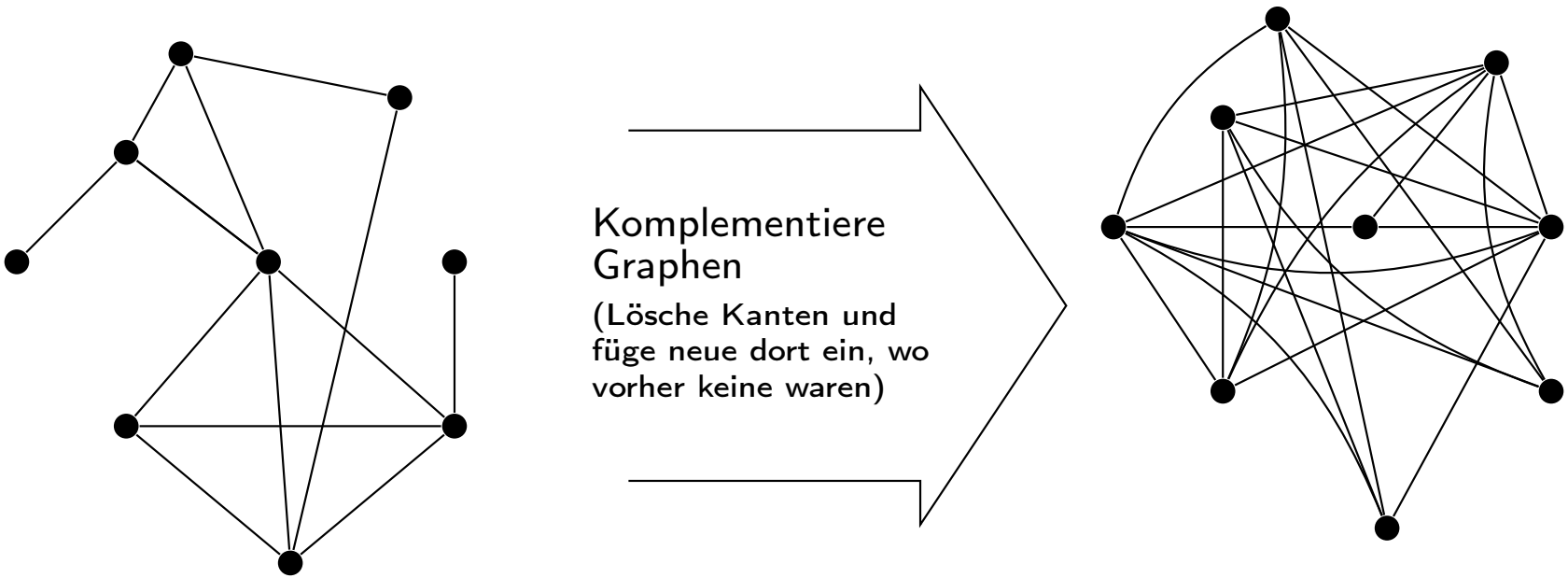
Polynomzeit-Reduktionen

Wir machen folgende Beobachtung



Polynomzeit-Reduktionen

Wir machen folgende Beobachtung



Polynomzeit-Reduktionen

Wir machen folgende Beobachtung

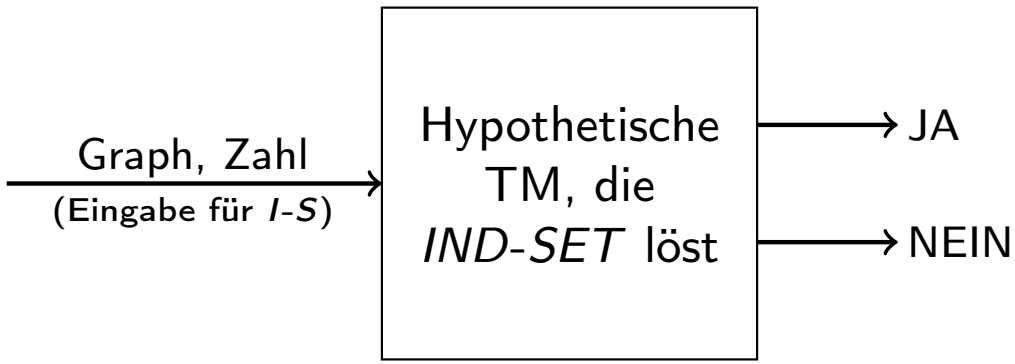
Clique der Grösse k wird zu IS der Grösse k

Polynomzeit-Reduktionen

Hypothetische TM, die *IND-SET* löst

- Angenommen, wir haben eine effiziente TM für *IND-SET*

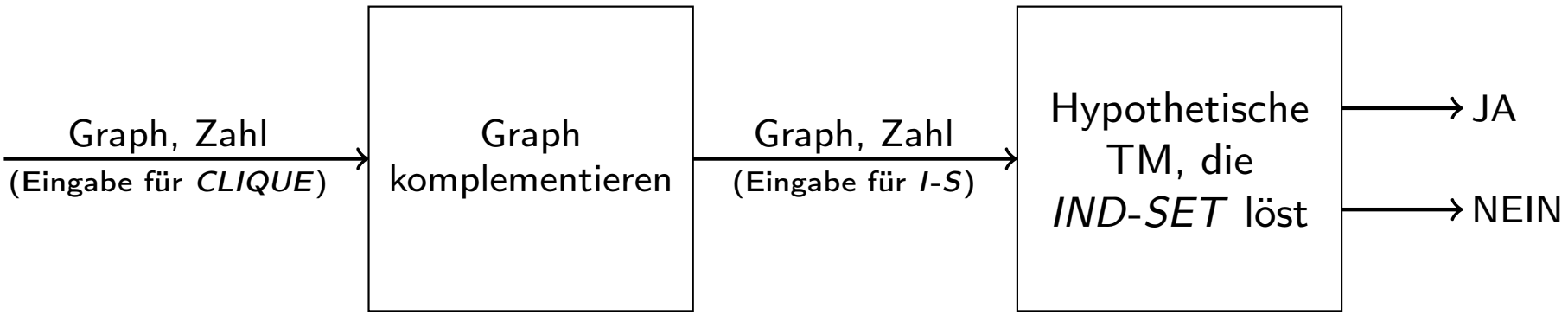
Polynomzeit-Reduktionen



- Angenommen, wir haben eine effiziente TM für *IND-SET*

⇨ Erhält Eingabe und gibt JA oder NEIN aus

Polynomzeit-Reduktionen

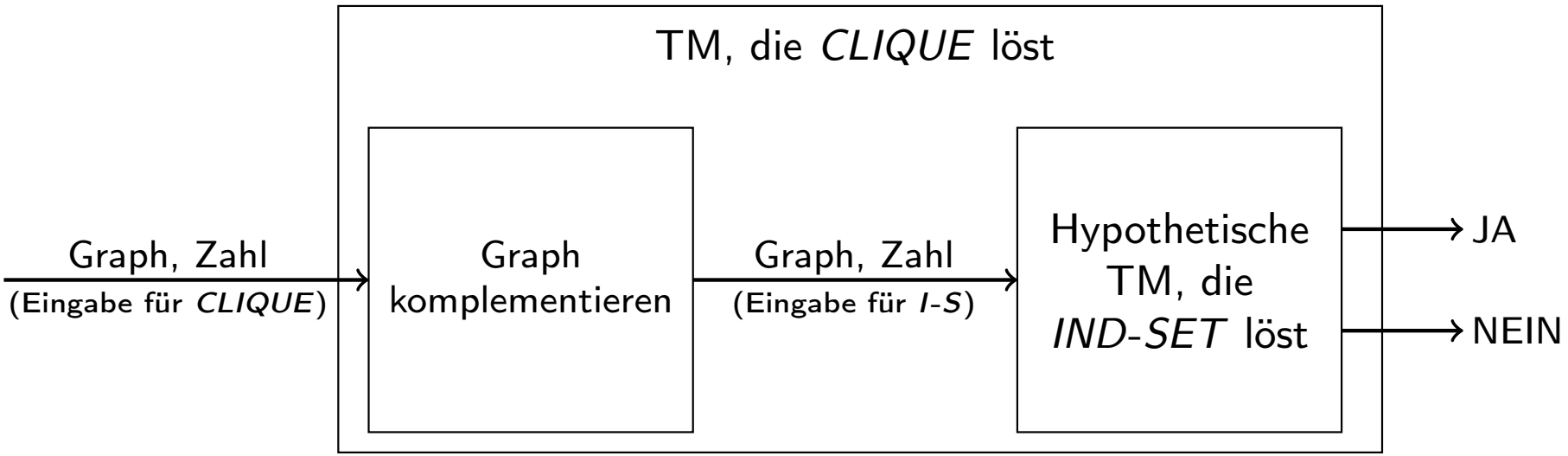


- Angenommen, wir haben eine effiziente TM für *IND-SET*

⇨ Erhält Eingabe und gibt JA oder NEIN aus

- Programm zum Komplementieren von Graphen vorschalten

Polynomzeit-Reduktionen



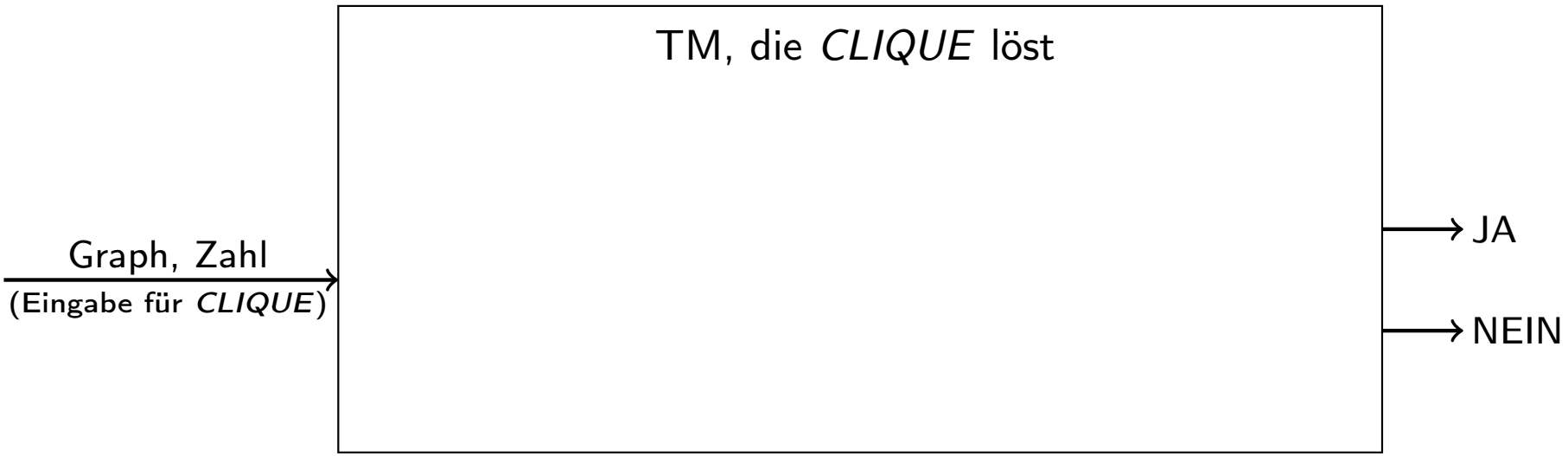
- Angenommen, wir haben eine effiziente TM für *IND-SET*

⇨ Erhält Eingabe und gibt JA oder NEIN aus

- Programm zum Komplementieren von Graphen vorschalten

⇨ Wir erhalten eine effiziente TM für *CLIQUE*

Polynomzeit-Reduktionen



- Angenommen, wir haben eine effiziente TM für *IND-SET*

⇨ Erhält Eingabe und gibt JA oder NEIN aus

- Programm zum Komplementieren von Graphen vorschalten

⇨ Wir erhalten eine effiziente TM für *CLIQUE*

Polynomzeit-Reduktionen

- Auf diese Weise konnten tausende $\mathcal{NP}$-vollständige Probleme identifiziert werden

Polynomzeit-Reduktionen

- Auf diese Weise konnten tausende $\mathcal{NP}$-vollständige Probleme identifiziert werden
- Wird jemals für eines von ihnen eine effiziente TM gefunden, existieren effiziente TM für sie alle

Polynomzeit-Reduktionen

- Auf diese Weise konnten tausende $\mathcal{NP}$-vollständige Probleme identifiziert werden
- Wird jemals für eines von ihnen eine effiziente TM gefunden, existieren effiziente TM für sie alle

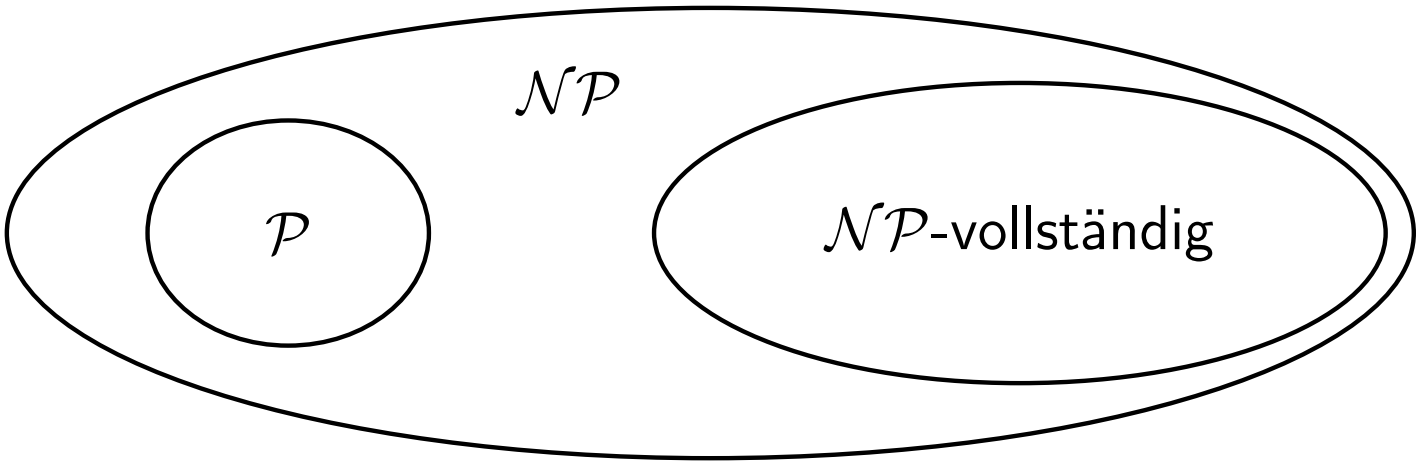
$\Rightarrow \mathcal{P} = \mathcal{NP}$

Polynomzeit-Reduktionen

- Auf diese Weise konnten tausende $\mathcal{NP}$-vollständige Probleme identifiziert werden
- Wird jemals für eines von ihnen eine effiziente TM gefunden, existieren effiziente TM für sie alle

$\Rightarrow \mathcal{P} = \mathcal{NP}$

- Derzeit gehen wir von folgender Beziehung aus



Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können

Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können
- Resultate über Entscheidungsprobleme lassen sich leicht auf „Suchprobleme“ erweitern

Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können
- Resultate über Entscheidungsprobleme lassen sich leicht auf „Suchprobleme“ erweitern
- Probleme, die gelöst werden können, können unterschiedlich gut gelöst werden

Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können
- Resultate über Entscheidungsprobleme lassen sich leicht auf „Suchprobleme“ erweitern
- Probleme, die gelöst werden können, können unterschiedlich gut gelöst werden
- „Effizient“ bedeutet in Polynomzeit

Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können
- Resultate über Entscheidungsprobleme lassen sich leicht auf „Suchprobleme“ erweitern
- Probleme, die gelöst werden können, können unterschiedlich gut gelöst werden
- „Effizient“ bedeutet in Polynomzeit
- Für viele Probleme kennen wir keine Polynomzeit-Algorithmen

Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können
- Resultate über Entscheidungsprobleme lassen sich leicht auf „Suchprobleme“ erweitern
- Probleme, die gelöst werden können, können unterschiedlich gut gelöst werden
- „Effizient“ bedeutet in Polynomzeit
- Für viele Probleme kennen wir keine Polynomzeit-Algorithmen

⇨ *HK*, *CLIQUE*, *IND-SET*, ...

Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können
- Resultate über Entscheidungsprobleme lassen sich leicht auf „Suchprobleme“ erweitern
- Probleme, die gelöst werden können, können unterschiedlich gut gelöst werden
- „Effizient“ bedeutet in Polynomzeit
- Für viele Probleme kennen wir keine Polynomzeit-Algorithmen

⇨ *HK*, *CLIQUE*, *IND-SET*, ...

- Dennoch können wir nicht beweisen, dass es nicht geht

Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können
- Resultate über Entscheidungsprobleme lassen sich leicht auf „Suchprobleme“ erweitern
- Probleme, die gelöst werden können, können unterschiedlich gut gelöst werden
- „Effizient“ bedeutet in Polynomzeit
- Für viele Probleme kennen wir keine Polynomzeit-Algorithmen

⇨ *HK*, *CLIQUE*, *IND-SET*, ...

- Dennoch können wir nicht beweisen, dass es nicht geht
- $\mathcal{P}$-versus-$\mathcal{NP}$-Problem offen seit Jahrzehnten

Zusammenfassung

- Es gibt Probleme, die algorithmisch nicht gelöst werden können
- Resultate über Entscheidungsprobleme lassen sich leicht auf „Suchprobleme“ erweitern
- Probleme, die gelöst werden können, können unterschiedlich gut gelöst werden
- „Effizient“ bedeutet in Polynomzeit
- Für viele Probleme kennen wir keine Polynomzeit-Algorithmen

⇨ *HK*, *CLIQUE*, *IND-SET*, ...

- Dennoch können wir nicht beweisen, dass es nicht geht
- $\mathcal{P}$-versus-$\mathcal{NP}$-Problem offen seit Jahrzehnten
- Eines der „Millennium-Probleme“ (1 000 000 USD Preisgeld)

Vielen Dank für die Aufmerksamkeit